Programming Normative Artifacts with Declarative Obligations and Prohibitions

N.A.M. Tinnemeier  
Utrecht University  
nick@cs.uu.nl

M. M. Dastani  
Utrecht University  
mehdi@cs.uu.nl

J-J. Ch. Meyer  
Utrecht University  
jj@cs.uu.nl

L. van der Torre  
University of Luxembourg  
leon.vandertorre@uni.lu

Abstract—Normative concepts (e.g., obligations and prohibitions) are seen as a suitable tool for regulating the observable behavior of software agents. An enforcement mechanism – a process that detects when a norm is active, detects violations of the norms and handles these violations – is crucial for the practical use of norms in the construction of multi-agent systems. We propose a programming language for programming normative artifacts in terms of “declarative” norms referring to a state of the world (as opposed to “procedural” norms only referring to actions), and introduce the operational semantics of the norm enforcement mechanism.

I. INTRODUCTION

The use of an explicit representation of normative concepts such as obligations and prohibitions is seen as a suitable tool to achieve or maintain the global objectives of a system in which multiple agents interact. Indeed, much literature can be found on the formal specification and verification of norms (e.g., [2], [3], [4], [5], [6]). The implementation of a norm enforcement mechanism – a process that is responsible for detecting when a norm is active, detecting violations and handling of these violations – is crucial for the operationalisation of norms [8]. Indeed, multiple research focuses on the issue of the practical implementation of norm enforcement mechanisms. Examples are the AMELI platform of Esteva et al. [9] and its extensions [10], [11], the normative framework of Cardoso et al. [14], law-governed interaction by Minsky et al. [12], and the norm enforcement mechanism of Fornara et al. [13]. Despite these developments we observe a gap between research on the construction of normative frameworks and multi-agent programming languages (e.g. [18]). We conjecture that the following two issues underly the root cause of this gap.

First, all of the above mentioned implementations (except for [14]) are primarily targeted at “procedural” norms specifying which actions agents ought or ought not perform and disregard the issue of expressing “declarative” norms related to a description of a desired state of affairs that should be brought about in an environment the agents interact with. Norms in existing frameworks thus typically take on the form of “ought-to-do” statements pertaining to actions rather than “ought-to-be” statements pertaining to the declarative description of a state. We argue that expressing declarative norms is also important, because:

1) If we can relate to actions only, writing norms to ensure a certain state is achieved might become a rather tedious task, especially when establishing it involves a multitude of actions and the endeavor of multiple agents. It is, for example, difficult to express a norm for a conference management system that at least two reviews should be received for each paper;

2) By only stating the actions the agents should perform to reach a desired situation we risk to limit their autonomy, because we leave them no choice in deciding how to reach it;

3) Related to the previous point, expressing declarative norms accords more with the concept of declarative goal as often used for modeling agents, viz. a description of a desirable situation. This conformity facilitates the internalization of norms by agents for reasoning with them (see for example [16]).

Second, current work on normative frameworks for regulating the behavior of agents is not endowed with an operational semantics [17] that allows for a direct implementation of an interpreter. An operational semantics allows us to evaluate a framework by formally studying the key properties it exhibits. Many agent-oriented programming languages are already investigated by means of an operational semantics (cf. [18]) and also doing this for the normative concepts contributes to our longer-term goal to study the properties of a multi-agent system regulated by norms.

This paper addresses these issues by providing an operational semantics of a norm enforcement mechanism for declarative obligations and prohibitions. In earlier research [15] we also addressed this issue, but in that approach the expression of norms is limited to counts-as rules, and conditional obligations and prohibitions with deadlines cannot be specified, restraining expressiveness and readability. In this paper we extend this language to include these concepts. The remainder of this paper is structured as follows. Section 2 gives some background and an intuitive explanation of our approach. Section 3 introduces the syntax of our language. Section 4 operationalises the norm enforcement mechanism by endowing the syntax with an operational semantics and shows some of the key properties it exhibits. Section 5 concludes this paper. As a running example we use a (simple) conference management system that supports authors and reviewers in the submission and review process.
II. AGENTS AND NORMATIVE ARTIFACTS

A multi-agent system, as we conceive it, is composed of a set of heterogeneous agents that interact with a collection of normative artifacts. By heterogeneity we mean that the agents are possibly implemented by different parties and nothing can be assumed about the behavior they will exhibit. Figure 1 shows a snapshot of a multi-agent system composed of agents that interact with a normative artifact. An artifact encapsulates a domain specific state and function, for instance, a database in which papers and reviews are stored and accompanying functions to upload them. The state of the artifact is modeled by a set of brute facts, taken from Searle [19]. The agents perform actions that change the brute state to interact with the artifact and exploit its functionality.

An important purpose of a normative artifact is to coordinate the behavior of its interactants and to guide them in interacting with it in a meaningful way. This is achieved through the normative component that is defined by the norm schemes, a set of conditional obligations and prohibitions. The norm schemes define under which conditions obligations and prohibitions should be created. For example, if a reviewer is assigned a paper to review, then an obligation to have uploaded a review for that paper is created. The condition of a norm scheme relates to the brute state of the artifact and whenever its condition is satisfied the artifact instantiates the obligation or prohibition belonging to it, hence the name norm instance.

The norm instances that are created out of the norm schemes are thus a set of active (unconditional) obligations and prohibitions. They specify which brute states should (not) be achieved. Usually, a norm instance is accompanied by a deadline (cf. [4], [6]). For example, an obligation to have reviewed a paper should be fulfilled before the notification phase starts. Intuitively, an obligation specifies which brute state should be established before a moment, whereas a prohibition specifies which brute state should be avoided until the deadline. Relying on the specification of deadlines as propositional formulae instead of time [6], in our approach, deadlines are expressed as a situation that can be entailed by the brute state. This gives us more freedom in specifying different notions of deadlines and allows us, for example, to relate to a situation that is established by the actions of other agents without the need to know if and when it will be established.

To motivate agents to abide by the norms, there is often a sanction associated to the violation of a norm (cf. [8], [7]). Note that an obligation is violated when the state denoted by it is not achieved before its deadline, whereas a prohibition is violated whenever the state denoted by it is established before its deadline. Each norm instance is equipped with a sanction that specifies a modification to the brute state. For example, an author that violates a prohibition to upload a paper of more than 15 pages is punished by an instant rejection of the paper by removing it from the database. The task of the artifact is to check the violation and fulfillment of norm instances and impose sanctions accordingly.

Previously, we explained that the conditions of norm schemes relate to the brute state. It should be noted that there are also norm instances that only rise in the sub-ideal situation in which some norm instance is violated (e.g. contrary-to-duty norms [3] which are a special case of this). For example, a reviewer that did not abide by its obligation to have reviewed a paper obliges the program chair to remind him/her of this task. We call a norm that is triggered by some condition of the brute state a primary norm, and a norm that rises due to the violation of a primary norm a secondary one. This resembles the notion of interlocking of norms as proposed in [7]. Information about which norm instances are violated is stored by the institutional facts (again as taken from Searle [19]). The conditions of norm schemes at level zero thus relate to brute facts, whereas a level \( k + 1 \) norm with \( k > 0 \) defines the obligations and prohibitions that arise in case of a violation of some norm(s) defined at level \( k \). Note that this implies that we assume the norms at level \( k+1 \) only refer to norms at level \( k \). We impose this restriction because we think it is conceptually sound and promotes the manageability and readability of the norm schemes.

Usually, a norm instance is directed at someone [7], but because the artifact’s interactants are not known beforehand, in expressing the norm schemes we rather refer to the roles that are played by agents instead of to the agents directly. For the present work we adopt a simple representation of roles and their enactment. A role is a label \( r \) that identifies it by a unique name. We model the institutional fact that an agent identified by \( a \) has enacted role \( r \) by a proposition \( \text{rea}(a,r) \). We assume operations provided by the normative artifact to enact and de-enact roles. These and a richer account of roles is specified elsewhere [20].

III. PROGRAMMING NORMATIVE ARTIFACTS

This section describes the syntax of the language by which normative artifacts are programmed. To program a normative artifact is to specify the roles agents can play, a set of facts
MAS = "Roles:" { (role) } | "Facts:" { (b-literal) } | "Effects:" { (effect) } | "Norm-level 0:" { (level-0) } | "Norm-level n:" { (level-n) };

(effect) = "pre" (pre) "post" (post) | "action" (action) "post" (post) |

(pre) = (b-literal) (r-literal) (pre) "post" (post) |

(post) = (b-literal) (r-literal) "post" |

(level-0) = "norm-0" (norm-0) |

(norm-0) = (label) | "condition" (cond-n) | "effect" (OP) |

(level-n) = (norm-n) |

(norm-n) = (label) "condition" (cond-n) |

(OP) = "OR" (b-test) | "AND" (b-test) |

(b-literal) = "not" (atom) |

(r-literal) = (atom) |

(b-test) = (atom) |

(atom) = "v-atom" |

(b-test) = "sanc" |

(r-test) = "ddln" |

(label) = "b-lit" |

(cond-n) = "b-test" | "norm-n" |

Code fragment 3.1 Conference management system.

Roles: chair, author, reviewer

Facts: phase (closed), id (0)

Effects:

{ rea (C, chair), phase (closed) } open (C)

{ not phase (closed), phase (abstracts) } uploadAbstract (A)

{ assign (A, PId), not id (PId), id (PId +1) } uploadPaper (A, PId)

{ assign (C, R, PId) } assign (R, PId)

{ phase (review), assigned (R, PId) } uploadReview (R, PId)

Figure 2. EBNF grammar of the programming language.

specifying the initial brute state, a set of effects specifying how the brute state evolves under the performance of actions and a set of norms. From this point on we use ⟨atom⟩ to denote an atom that starts with a lowercase letter. We disjointly divide the set of atoms into ⟨b-atom⟩s that model the brute state, ⟨v-atom⟩s which are atomic formulae of the form viol(l) to model the fact that a norm with label l has been violated, and ⟨r-atom⟩s which are special atomic formulae of the form rea(a, r) to model the fact that an agent identified by a has enacted role r (rea is short for role enacting agent). The ⟨b-atom⟩s pertain to the brute state of the artifact, whereas the ⟨v-atom⟩s and ⟨r-atom⟩s pertain to the institutional state of the artifact. Actions are denoted by ⟨action⟩, a special atomic formula of which the predicate denotes the name of the action and the terms its arguments. We assume that the first argument is always the identifier of the agent performing it. Finally, ⟨role⟩ denotes the unique identifier of a role agents can play and ⟨label⟩ is a first-order atom with constants and variables used to uniquely identify norms (their use will be explained later on).

We explain the intuitive semantics of the syntactical constructs as defined in figure 2 by our conference management system example. The system goes through six subsequent phases, namely the phase in which: the system is closed, abstracts can be uploaded, papers are submitted, submissions are reviewed, the reviews are collected, and finally, the phase in which authors are notified. The code that implements the system is listed in code fragments 3.1 and 3.2.

The roles are specified in line 1. Agents can thus play the role of author, reviewer and chair. For this system we assume that only one agent plays the role of chair. Again, the policy and functionality for role enactment and de-enactment is beyond the scope of this paper. We assume that when an agent a has successfully enacted a role r a fact rea(a, r) is asserted to the institutional state, and this fact is removed upon de-enactment.

Information about, amongst others, abstracts and papers that are uploaded by authors, papers that are assigned during the review phase. The code that implements the CMS is (incompletely) defined in lines 3-23. The chair can open the submission system which will put it in the abstract phase. In this phase an author can perform an action to upload an abstract and is assigned a unique id. Then, in the submission phase authors can upload their papers. The fact that a paper with id PId belonging to author A is stored in the database is modelled by a fact paper (A, PId) . Note that papers can only be uploaded when an abstract was uploaded. Moreover, the program chair is able to divide those papers amongst the reviewers, and a reviewer can upload the review of its assigned papers during the review phase.

The conference management system has some behavioral expectations about its participants, which are expressed by the norm schemes, that is, conditional obligations and prohibitions. A conditional obligation is expressed as a labelled tuple of the form l: ⟨c, O(x), d, s⟩ with the intuitive reading “if condition c holds then there is an obligation to establish the brute state denoted by x before deadline d, otherwise sanction s will be imposed”. A conditional
prohibition is expressed as a tuple \( l : (c, F(x), d, s) \) that can be intuitively read as “if condition \( c \) holds then it is forbidden to establish the brute state denoted by \( x \) before deadline \( d \), otherwise sanction \( s \) will be imposed.” Norms are equipped with a parameterized label that is used to uniquely identify them and to keep track of which of them are violated. Remember that we distinguish between primary norms (also called level zero norms) that are triggered because of a particular condition of the brute state and secondary norms that are triggered due to the violation of another norm. The condition of primary norms thus relates to brute facts, whereas the condition of norms at level one and beyond relates to violation facts as stored in the institutional state. Note that the condition of both type of norms can refer to \( \text{rea} \) facts to associate them with roles and their players. We acknowledge that norm schemes can contain more components as, for example, explained in [7]. For this paper we limit ourselves to the elementary components that are needed to explain the semantics of the enforcement mechanism. Other components such as addressees, beneficiaries and rewards can be incorporated.

The level zero norm schemes of the CMS are listed in code fragment 3.2 (lines 24-41). The first norm scheme expresses that uploaded papers should not exceed the page limit of 15 pages. Suppose an author, say \( \text{jane} \), has uploaded an abstract and has been assigned id. 547. As soon as the chair puts the system in the submission phase the condition is satisfied and the norm scheme is instantiated into a norm instance pertaining to a prohibition \( F(\text{page\_size}(547) > 15) \) stating that \( \text{jane} \)’s paper is not allowed to exceed 15 pages. This prohibition stays into effect during the whole submission phase, i.e. until the review phase starts. Hence, its deadline \( \text{phase\_review} \). A violation is detected as soon as \( \text{jane} \) uploads a paper of more than 15 pages, which is sanctioned by an instant rejection of the paper. Note that

\[
\text{code fragment 3.2 Conference management system cont.}
\]

\[
\begin{align*}
\text{Norm – level 0:} & \\
\text{page\_size():} & 24 \\
\langle \text{phase\_submission}() \text{ and abstract}(A,PId) \rangle & 25 \\
\langle F(\text{page\_size}(PId)) > 15 \rangle & 26 \\
\text{phase}() & 27 \\
\langle \text{not paper}(A,PId) \rangle & 28 \\
\text{review\_due():} & 29 \\
\langle \text{phase}(\text{review}) \text{ and assigned}(R,PId) \rangle & 30 \\
\langle O(\text{review}(R,PId)) \rangle & 31 \\
\langle \text{phase}(\text{collect\_reviews}) \rangle & 32 \\
\text{blacklist}(R) & 33 \\
\text{minimum\_reviews}(PId): & 34 \\
\langle \text{phase}(\text{submission}) \text{ and paper}(PId) \rangle & 35 \\
\langle O(\text{nr\_reviews}(PId)) >= 2 \rangle & 36 \\
\text{minimum\_reviews}(PId) & 37 \\
\langle \text{phase}(\text{collect\_reviews}) \rangle & 38 \\
\text{blacklist}(C) & 39 \\
\text{viol\_minimum():} & 40 \\
\langle \text{viol}(\text{minimum\_reviews}(PId)) \text{ and rea}(C,\text{chair}) \rangle & 41 \\
\langle O(\text{nr\_reviews}(PId)) >= 2 \rangle & 42 \\
\text{minimum\_reviews}(PId) & 43 \\
\langle \text{phase}(\text{notification}) \rangle & 44 \\
\text{blacklist}(C) & 45 \\
\text{viol\_minimum():} & 46 \\
\langle \text{viol}(\text{minimum\_reviews}(PId)) \text{ and rea}(C,\text{chair}) \rangle & 47 \\
\langle O(\text{nr\_reviews}(PId)) >= 2 \rangle & 48 \\
\end{align*}
\]

![Figure 3. The possible evolutions of an obligation (left) and a prohibition (right). In each program state the institutional state is shown on the first line, the brute on the second and the norm instances on the third line.](image)

the sanction is imposed on the brute state. The second norm scheme states that a reviewer is obliged to have uploaded its assigned reviews before the reviews are collected. This obligation becomes active as soon as a reviewer is assigned a paper and stays into effect until it is either fulfilled (the review has been uploaded) or it is violated (the review has not been uploaded before the deadline). The third norm specifies that by the end of the reviewing phase there should be at least two reviews per paper. Note that this norm does not have a sanction associated to it, which is denoted by \( \top \).

A violation of the latter norm, however, does give rise to a new obligation for the chair to write a review for this paper. This is expressed by the norm scheme of the level one norms (lines 43-48). Suppose, for example, that when the review phase has ended still only one review for paper 547 has been received. Then the obligation \( O(\text{nr\_reviews}(547)) >= 2 \) is violated which is marked by the assertion of a fact \( \text{viol}(\text{minimum\_reviews}(547)) \) to the institutional state. This will trigger the norm scheme of lines 44-48 and will instantiate an obligation for the chair to have uploaded a review for paper 547 before the notification phase starts. Note how parameters of labels are used to pass information between norm schemes at different levels. It should be emphasised that the parameters of the labels are formal output parameters that become actual parameters upon instantiation of the norm scheme.

To recapitulate, a norm scheme instantiates a norm instance (obligation or prohibition) when its condition is satisfied. We write a norm instance as a tuple \( (l,c,\mathbb{P}(x),d,s) \) with \( \mathbb{P} \) either \( \Omega \) (obligation) or \( \mathbb{F} \) (prohibition). Once instantiated the behavior of an obligation and a prohibition differs as illustrated by figure 3. An obligation is removed (de-instantiated) by the system when it is fulfilled or violated. An obligation is fulfilled whenever the obliged situation denoted by \( x \) is established before the deadline \( d \) is entailed by the brute state, whereas it is violated whenever \( x \) is not achieved until \( d \) is entailed by the brute state. This semantics of an obligation resembles the one presented in [6]. A prohibition is removed only when its deadline \( d \) holds in the brute state irrespective of whether it is violated. A prohibition is violated when the forbidden situation as denoted by \( x \) holds
in the brute state before the deadline has passed. A violation of an obligation or prohibition is sanctioned by imposing the sanction $s$ on the brute state.

IV. Executing Normative Artifacts

In the previous section we have defined the syntax of the language by which normative artifacts can be specified and have shown an example of such an implementation. In this section we explain how normative artifacts are executed by endowing the syntax with an operational semantics [17], which describes the behavior of a programming language in terms of transitions between program configurations. A configuration describes a state of the program and a transition is a transformation of one configuration $\gamma$ into another configuration $\gamma'$, denoted by $\gamma \rightarrow \gamma'$. The transitions that can be derived for a programming language are defined by a set of derivation rules of the form $\frac{P}{\gamma \rightarrow \gamma'}$ with the intuitive reading that transition $\gamma \rightarrow \gamma'$ can be derived in case premise $P$ holds. An execution trace in a transition system is then a sequence of configurations that can be generated by applying transition rules to an initial configuration. An execution thus shows a possible behavior of the system at hand. All possible executions for an initial configuration show the complete behavior. The notion of an execution trace is formally defined below.

Definition 1 (Execution Trace): An execution in a transition system $T$ is a (possibly infinite) sequence of transitions $\gamma_0 \rightarrow \gamma_1 \rightarrow \ldots \rightarrow \gamma_n$ (often written as $\gamma_0 \rightarrow^* \gamma_n$) such that for each for $0 \leq i < n$ a derivation $\gamma_i \rightarrow \gamma_{i+1}$ can be made in $T$. □

To understand the execution of normative artifacts, we need to define the configuration for a normative artifact and the transition system that explains how these configurations may evolve. The configuration of a normative artifact consists of a set of brute literals pertaining to the domain specific state, a set of institutional literals containing information about the violation of norms and the enactment of roles, a set of norm schemes (conditional obligations and prohibitions) and a set of active obligations and prohibitions, i.e. the instantiated norm schemes. The actions to interact with the artifact are initiated by agents outside the scope of the artifact. We assume a process that receives and handles actions directed at the artifact and stores them in an event list. This architecture accords with a distribution of agents and artifact.

Before providing the formal definition of a configuration, we first define the notion of instantiating a norm. As explained before, a norm instance is instantiated from its norm scheme when its condition is derivable from the brute and institutional state for some substitution of its formal parameters. Instantiating a norm scheme is then to apply this substitution on it resulting in a norm instance. Let $ns = l(\overline{v}) : \langle c(\overline{v}), P(x(\overline{v})), d(\overline{v}), s(\overline{v}) \rangle$ be a norm scheme with $P$ either an obligation $O$ or prohibition $F$ and $\overline{v}_1, \ldots, \overline{v}_5$ the sets of variables occurring in the formulae. Then, in what follows, we use a function $\text{inst}(ns, \theta)$ that instantiates a norm instance from a norm scheme $ns$ given a formal substitution $\theta$ for the variables:

$$\text{inst}(ns, \theta) = \{ l(\overline{v}), c(\overline{v}), P(x(\overline{v})), d(\overline{v}), s(\overline{v}) \} \theta$$

The configuration of a normative artifact is defined as follows.

Definition 2 (Normative Artifact): A (normative) artifact configuration is a tuple $(\delta, \sigma_b, \sigma_e, \epsilon, \Delta_0 \cdot \Delta_n)$ with:

- $\Delta_i$ for $0 \leq i \leq n$ a set of level $i$ norm schemes by which the normative component of the artifact is defined. We write $\Delta$ to denote the set of all norm schemes, i.e. $\Delta = \bigcup_{i=0}^n \Delta_i$.
- $\delta \subseteq \{ \text{inst}(ns, \theta) \mid ns \in \Delta$ and $\theta$ a ground substitution of the variables $\}$, i.e. a set of ground norm instances;
- $\sigma_b$ a consistent set of ground literals, the brute state;
- $\sigma_e$ a set of ground atoms $\text{viol}(l)$, denoting the violation of a norm labeled $l$, and ground literals built of $\text{real}(a, r)$ atoms denoting that agent $a$ has enacted role $r$;
- $\epsilon$ a list of $\langle \text{atom} \rangle$s, the event base that stores the actions that are received by the artifact. We write $e.e$ to denote a list with head $e$ and tail $e$ and we write $e.e$ to mean that element $e$ is appended at the end of the list.

A configuration $(\emptyset, \sigma_b, \emptyset, \emptyset, \Delta_0 \cdot \Delta_n)$ specified by a program with $n$ norm levels and a brute state that is characterized by the facts component is called an initial artifact configuration. □

Note that we assume the norm instances to be ground, i.e. no variables occur in them. To ensure a norm instance to be ground, we demand each variable that occurs in the norm scheme also to occur in the condition. More formally, for all norm schemes $l(\overline{v}) : \delta(\overline{v}), P(x(\overline{v})), d(\overline{v}), s(\overline{v})$ we demand that $\overline{v} \cap \overline{v}_1 \cap \overline{v}_2 \subseteq \overline{v}_3$. Without this restriction unground obligations and prohibitions might be instantiated raising the question whether the variables occurring in them are existentially or universally quantified. This question and a possible extension of the language to include quantifiers is left for future research.

Henceforth, we write $(l, c, P(x), d, s)$ for a norm instance in which all the formulae $l$, $c$, $x$, $d$ and $s$ are ground. For the sake of representation, given a norm instance $ni = (l, c, P(x), d, s$) we define the functions $\text{label}(ni)$ to evaluate to the label $l$, $\text{cond}(ni)$ to evaluate to the condition $c$, $\text{op}(ni)$ to evaluate to the obligation or prohibition $P(x)$, $\text{dead}(ni)$ to evaluate to the deadline $d$, and $\text{sanc}(ni)$ to evaluate to the sanction $s$ of norm instance $ni$. We overload these functions in the expected manner to operate on norm schemes also. Moreover, we define the following auxiliary functions to operate on sets of norm instances $S$:

$$\text{Label}(S) = \{ l \mid (l, c, P(x), d, s) \in S \}$$

$$\text{Sanc}(S) = \{ s \mid (l, c, P(x), d, s) \in S \}$$
Next, we define the transition rules that that specify how a normative artifact changes under the performance of actions as performed by the agents. Responding to an action is not only a matter of accommodating its effect to the brute state; a change in the brute state might trigger new obligations and prohibitions. Moreover, in this new brute state some obligations and prohibitions might become fulfilled or even violated. It is the artifacts’s task to compute newly triggered, fulfilled and violated obligations and prohibitions. To be able to verify if a scheme should trigger and a norm instance and prohibitions. Moreover, in this new brute state some norms are triggered under the performance of actions respectively the set of violated obligations and prohibitions passed. A prohibition is thus violated in a state when the deadline has passed, but nevertheless the prohibition is violated or achieved we need to define an entailment condition of the norm schemes on a set of brute literals.

As we shall see later on, an obligation is only in effect when it has not been achieved yet and its deadline has not passed. So, when there is an obligation \( \langle l, c, O(x), d, s \rangle \) in a state and the deadline \( d \) has passed, but \( x \) has not been achieved in this state then this obligation is violated. Prohibitions are only in effect when their deadline has not passed. A prohibition is thus violated in a state when the deadline cannot be entailed, but nevertheless the prohibition is violated or achieved we need to define an entailment condition of the norm schemes on a set of brute literals.

Recall that triggering norms boils down to evaluating the condition of the norm schemes on a set of brute literals (for level 0 norms) or institutional literals (level 1 norms and up) and instantiating them. For this purpose we define the function \( T(S,X) \) that evaluates to the set of triggered norm instances that are to be instantiated given a set of norm schemes \( S \) and a set of literals \( X \):

\[
T(S,X) = \{ \text{inst}(ns, \theta) \mid ns \in S, X \models \text{cond}(ns)\theta \}
\]

For ground substitution \( \theta \)

As we shall see later on, an obligation is only in effect when it has not been achieved yet and its deadline has not passed. So, when there is an obligation \( \langle l, c, O(x), d, s \rangle \) in a state and the deadline \( d \) has passed, but \( x \) has not been achieved in this state then this obligation is violated. Prohibitions are only in effect when their deadline has not passed. A prohibition is thus violated in a state when the deadline cannot be entailed, but nevertheless the prohibition is violated or achieved we need to define an entailment condition of the norm schemes on a set of brute literals.

The following functions \( V_O(S,X) \) and \( V_P(S,X) \) evaluate to respectively the set of violated obligations and prohibitions given a set of norm schemes \( S \) and a set of brute literals \( X \):

\[
V_O(S,X) = \{ \langle l, c, O(x), d, s \rangle \in S \mid X \models d, X \not\models x \}
\]

\[
V_P(S,X) = \{ \langle l, c, F(x), d, s \rangle \in S \mid X \not\models d, X \models x \}
\]

The process of determining which obligations and prohibitions are triggered in a certain brute state and determining which norm instances are violated is explained by two auxiliary transition rules. These rules are used in the definition of the main rule that defines the transitions pertaining to the artifact’s norm enforcement mechanism. To discern them from this transition we write them as \( \leftarrow \) instead of \( \rightarrow \). The process of triggering and determining violations proceeds by different steps. During each step the triggered norms are determined for only one norm level at a time. To mark which set of norm schemes \( \Delta_i \) pertaining to the level \( i \) norms is next for the process of triggering and determining violations we annotate it with an arrow \( \downarrow \), i.e. \( \downarrow \Delta_i \) means that \( \Delta_i \) is next. The first transition rule then shows the process of triggering and determining violations based on the primary norm schemes as defined by \( \Delta_0 \).

**Rule 1:** Let \( \langle \delta, \sigma_b, \sigma_i, \epsilon, \Delta_0 \cdots \Delta_n \rangle \) be an artifact configuration, then the rule for normatively assessing this artifact based on the level 0 norms is defined as:

\[
\delta' = \delta \cup T(\Delta_0, \sigma_b) \quad V = V_O(\delta', \sigma_b) \cup V_P(\delta', \sigma_b)
\]

\[
\sigma' = \sigma_i \cup \{ \text{viol} \mid l \in \text{Label}(V) \}
\]

\[
\langle \delta, \sigma_b, \sigma_i, \epsilon, \Delta_0 \Delta_1 \cdots \Delta_n \rangle \leftarrow \langle \delta', \sigma_b, \sigma_i', \epsilon, \Delta_0 \Delta_1 \cdots \Delta_n \rangle
\]

Recall that norms at any level may refer to real propositions, which are stored in the institutional state. Therefore, in determining which norm schemes are triggered also the institutional state is used. Observe that the institutional state is extended to contain the violation facts of all the norm instances that are violated. This information is used in determining the triggered norms of level one and beyond. Also note that the focus of the norm schemes level is now set just behind \( \Delta_0 \) giving focus to \( \Delta_1 \) (if present). The following transition rule specifies the process of triggering and determining violations for the norm scheme levels greater than zero. This rule is similar to the rule for primary norms as defined above, with the exception that the triggered norms are now based on the norm schemes of level \( j \) that is removed when the deadline is passed, i.e. \( x \) is removed.

**Rule 2:** Let \( \langle \delta, \sigma_b, \sigma_i, \epsilon, \Delta_0 \cdots \Delta_n \rangle \) be an artifact configuration, then the rule for normatively assessing this artifact based on the level \( j \) norms with \( j > 0 \) is defined as:

\[
\delta' = \delta \cup T(\Delta_j, \sigma_i) \quad V = V_O(\delta', \sigma_b) \cup V_P(\delta', \sigma_b)
\]

\[
\sigma' = \sigma_i \cup \{ \text{viol} \mid l \in \text{Label}(V) \}
\]

\[
\langle \delta, \sigma_b, \sigma_i, \epsilon, \Delta_0 \cdots \Delta_0 \Delta_1 \cdots \Delta_n \rangle \leftarrow \langle \delta', \sigma_b, \sigma_i', \epsilon, \Delta_0 \Delta_1 \cdots \Delta_n \rangle
\]

Observe that the focus is increased by one level each time this rule is applied and note that this rule is not applicable in case \( \Delta_0 \cdots \Delta_j \downarrow \) which means that focus has moved beyond the last level of norm schemes. The following transition rule specifies the handling of an action by the artifact. This process is a matter of determining the effect of the action by means of the effect rules and normatively assessing the new brute state that results from the performance of this action. In normatively assessing the brute state, we only determine which norms instances are violated in this brute state, that is, previous violations are not remembered. When all levels of norm schemes are consecutively considered in the process of triggering, the norm instances that are no longer in effect are removed from the system and, finally, sanctions are applied. As explained before, an obligation \( \langle l, O(x), d, s \rangle \) is removed when the deadline has passed or when the state denoted by it is established, i.e. when the brute state entails \( d \) or \( x \). A prohibition \( \langle l, F(x), d, s \rangle \) is removed when the deadline is passed, i.e. \( d \) is entailed by the brute state. In what follows we assume a consistency preserving update operator \( \oplus \).
Rule 3: Let \( \langle \delta, \sigma_b, \sigma_1, \alpha, \epsilon, \Delta_0 \cdots \Delta_n \rangle \) be a normative artifact and \( E(\alpha, X) \) be a function that determines the effect of the performance of action \( \alpha \) given a set of literals \( X \), then the multi-agent transition rule for external actions is defined as:

\[
\sigma'_1 = E(\alpha, \sigma_1) \quad \sigma'_1 = \sigma_1 \cdot \{ \text{violated} | \text{violated} \in \sigma_1 \}
\]

\[
\delta'_n, \sigma'_n, \sigma'_0, \alpha, \epsilon, \Delta_0 \cdots \Delta_n \rightarrow (\delta_n, \sigma'_n, \sigma'_0, \alpha, \epsilon, \Delta_0 \cdots \Delta_n)
\]

where \( \delta'''' = \delta'''' \setminus \{ (l, F(x), d, s) | (l, F(x), d, s) \in \delta' \text{ and } \sigma'_0 = \emptyset \} \)

\[
\delta''' = \delta'' \setminus \{ (l, O(x), d, s) | (l, O(x), d, s) \in \delta' \text{ and } \sigma'_0 = \emptyset \} \text{ or } \sigma'_0 = \emptyset = d \}
\]

\[
\sigma''' = \sigma'_0 \triangle V(\delta''', \sigma'_0) \cup V(\delta', \sigma'_0)
\]

The application of a sanction might establish/undo the action \( d \) and \( x \) components of a norm instance \( (l, c, \mathbb{P}(x), d, s) \). Consider for example, a configuration \( \gamma \) such that \( (l, c, O(x), d, s) \in \delta \) and suppose that after applying the effect of an action \( x \) \( \sigma'_0 = \emptyset \). Then from the definition of rule 3 it follows that \( (l, c, O(x), d, s) \) will be removed (it is achieved). Now, suppose that some other norm is violated which has as sanction not \( x \). Then after applying this sanction \( \sigma'''' \neq x \), but \( (l, c, O(x), d, s) \) is still absent. We chose to only consider the effects of actions in determining violations and achievements. We intend to explicitly separate the sanctioning mechanism from the monitoring mechanism in the future. To prove that in the operational semantics as defined above the interpretation of obligations and prohibitions is indeed the one as explained in section 3 we need to add a condition to rule 3 s.t. all norm instances in \( \delta' \) are mutually non-interfering, i.e. we add:

\[
\text{noninterfering}(\delta', \sigma'_0) = \forall (l, c, \mathbb{P}(x), d, s), (l', c', \mathbb{P}(x'), d', s') \in \delta' : \sigma'_0 = d' \text{ iff } \sigma'_0 \uplus s = d', \text{ and } \sigma'_0 = x' \text{ iff } \sigma'_0 \uplus s = x'
\]

A trace is non-interfering if it is generated by rule 3 enriched with the above condition.

In the sequel, referring to the application of rule 3, we use \( \sigma_{b,j} (\sigma''', \sigma''^j) \) to refer to the brute state (brute state after applying the effect of an action, brute state after applying sanctions) of configuration \( \gamma_j \). Similar notations are used for the other components of \( \gamma \). The first proposition shows that under the assumption of interference-freeness an obligation persists as long as it is not achieved and it is still before the deadline. The second shows the same case for a prohibition that will persist as long as its deadline is not there.

Proposition 1: Let \( \gamma_0 \) be an artifact configuration s.t. \( (l, c, O(x), d, s) \in \delta_0 \). Then for every non-interfering trace \( \gamma_0 \rightarrow^* \gamma_n \) with \( \sigma_{b,j} \neq x \) and \( \sigma_{b,j} \neq d \) for \( 0 \leq j < n \) it holds that \( (l, c, O(x), d, s) \notin \delta_n \).

Proof: For \( \delta_0 \) we have \( (l, c, O(x), d, s) \notin \delta_0 \) by definition. We have to prove that if \( (l, c, O(x), d, s) \in \delta_k \), then \( (l, c, O(x), d, s) \in \delta_{k+1} \). Suppose \( (l, c, O(x), d, s) \in \delta_k \). The application of rule 3 ensures that \( (l, c, O(x), d, s) \) is not removed because 1) \( \delta_k \subseteq \delta_k'' \), 2) \( \delta_k'' \) still contains \( (l, c, O(x), d, s) \) as \( \sigma_{b,j} \neq x \) and \( \sigma_{b,j} \neq d \) (by assumption) and the fact that the trace is non-interfering, and 3) \( \delta_k'' \) is the same as \( \delta_k'' \) except that only (forbidden) norms of the form \( (l', c', F(x'), d', s') \) may have been removed. We conclude that \( (l, c, O(x), d, s) \in \delta_k'' \) and rule 3 ensures that \( \delta_{k+1} = \delta_{k+1}'' \) such that we can conclude that \( (l, c, O(x), d, s) \notin \delta_{k+1} \).

Proposition 2: Let \( \gamma_0 \) be an artifact configuration s.t. \( (l, c, F(x), d, s) \in \delta_0 \). Then for every non-interfering trace \( \gamma_0 \rightarrow^* \gamma_n \) with \( \sigma_{b,j} \neq d \) for \( 0 \leq j < n \) it holds that \( (l, c, F(x), d, s) \notin \delta_n \).

Proof: The proof proceeds by a similar reasoning as that of proposition 1. Note that by the definition of rule 3 a prohibition is only removed in case \( \sigma_{b,j} = d \).

The next two propositions pertain to the case in which a norm instance is violated. A prohibition is violated when the state denoted by it is achieved before the deadline, whereas an obligation is violated when the state denoted by it is not achieved before the deadline.

Proposition 3: Let \( \gamma_0 \) be an artifact configuration s.t. \( (l, c, O(x), d, s) \in \delta_0 \). Then for every non-interfering trace \( \gamma_0 \rightarrow^* \gamma_n \) s.t. \( \sigma_{b,j} \neq x \) and \( \sigma_{b,j} \neq d \) for \( 0 \leq j < n \) and \( \sigma_{b,n} \neq x \) and \( \sigma_{b,n} \neq d \) it holds that a) \( \sigma_{i,n} = \text{viol}(l) \) and b) \( (l, c, O(x), d, s) \notin \delta_n \).

Proof: Assume a derivation \( \gamma_0 \rightarrow^* \gamma_{k+1} \) s.t. \( (l, c, O(x), d, s) \in \delta_k \) and \( \sigma_{b,j} \neq x \) and \( \sigma_{b,j} \neq d \) for \( 0 \leq j < k+1 \) and \( \sigma_{b,k+1} \neq x \) and \( \sigma_{b,k+1} \neq d \) by assumption) and the fact that the trace is non-interfering. 3) \( \sigma_{i,k} = \text{viol}(l) \) because further applications of rule 2 will only add violation facts. Rule 3 assures that \( \sigma_{i,k+1} = \sigma_{i,k}'' \), we thus conclude \( \sigma_{i,k+1} = \text{viol}(l) \).

The application of rule 3 ensures that \( (l, c, O(x), d, s) \notin \delta_{k+1} \) because 1) \( (l, c, O(x), d, s) \) will be removed from \( \delta_k'' \) because \( \sigma_{b,k+1} \neq d \) (by assumption) and the fact that the trace is non-interfering, and 2) \( \delta_k'' \subseteq \delta_k'' \) (only norms may have been removed). Rule 3 assures that \( \delta_{k+1} = \delta_{k+1}'' \), we thus conclude \( (l, c, O(x), d, s) \notin \delta_{k+1}'' \).

Proposition 4: Let \( \gamma_0 \) be an artifact configuration s.t. \( (l, c, F(x), d, s) \in \delta_0 \). Then for every non-interfering trace \( \gamma_0 \rightarrow^* \gamma_n \) s.t. \( \sigma_{b,j} \neq d \) for \( 0 \leq j < n \) and \( \sigma_{b,n} \neq x \) and \( \sigma_{b,n} \neq d \) it holds that \( \sigma_{i,n} = \text{viol}(l) \) and \( (l, c, F(x), d, s) \in \delta_n \).

Proof: Based on a similar reasoning as the proof of proposition 3.

This last proposition shows the relation between obligation and prohibition. It proves that a violation is inevitable in case there is an obligation to reach a state \( x \) while during the whole period until the deadline associated with this obligation it is forbidden to establish \( x \).
Proposition 5: Let $\gamma_0$ be an artifact configuration with $(l, c, O(x), d, s) \in \delta_0$ and $(l', c', F(x), d', s') \in \delta_0$ and $\gamma_0 \rightarrow^* \gamma_n$ be an non-interfering trace. If $\sigma_{b, n} \models d$ and $\sigma_{b, j} \not\models d'$ for $0 \leq j \leq n$ then $\exists 0 \leq k \leq n : \sigma_{i, k} \models viol(l)$ or $\sigma_{i, k} \models viol(l')$.

Proof: Along the trace $x$ is either established or not:

- $\exists 0 \leq j \leq n : \sigma_{b, j} \models x$. Then from proposition 3 it follows that $\sigma_{i, j} \models viol(l')$;
- $\sigma_{b, j} \not\models x$ for all $0 \leq j \leq n$. Then from proposition 4 we have that $\sigma_{i, n} \models viol(l)$.

Thus it holds that $\exists 0 \leq k \leq n : \sigma_{i, k} \models viol(l)$ or $\sigma_{i, k} \models viol(l')$.

V. CONCLUSION AND FUTURE WORK

In this paper we presented a programming language for implementing normative artifacts directly in terms of “declarative” obligations and prohibitions referring to a state of the world as opposed to “procedural” obligations and prohibitions directly referring to actions. We explained the mechanism that is responsible for enforcing the norms in terms of an operational semantics [17] which allows for an almost direct implementation of an interpreter. Moreover, we investigated some of the key properties of our framework, in particular, the semantics of obligations and prohibitions.

We see the following directions for future research. To gain more insight in the usefulness of the concepts proposed, we are in the process of building an interpreter. Further, as for now, the norms cannot be muted at runtime. To promote flexibility they should be. For example, a norm can be made by an agent, as legislators in a legal system. Likewise, if the system observes that a norm is often violated, then apparently the norm does not work as desired, and it undermines the trust of the agents in the normative system, so the system can suggest that the agents can vote whether to retract or change the norm. Future research will focus on (amongst others) these issues.

Acknowledgements Work partially supported by the CoCoMAS project funded through the Dutch Organization for Scientific Research (NWO). We like to thank the anonymous reviewers, Huib Aldewereld and Frank Dignum for their useful comments and discussions.

REFERENCES