Agent Reasoning for Norm Compliance

A Semantic Approach

M. Birna van Riemsdijk[*]
Delft University of Technology
Delft, The Netherlands
m.b.vanriemsdijk@tudelft.nl

Louise Dennis
University of Liverpool
Liverpool, UK
l.a.dennis@liverpool.ac.uk

Michael Fisher
University of Liverpool
Liverpool, UK
mfisher@liverpool.ac.uk

Koen V. Hindriks
Delft University of Technology
Delft, The Netherlands
k.v.hindriks@tudelft.nl

ABSTRACT

A system of autonomous agents may exhibit undesirable or ineffective behavior if no form of regulation is imposed. Norms, describing how agents should ideally behave, can be used to address this issue if agents are able to reason about norms and adapt their behavior to comply with them (if they choose to do so). Assuming that which norms will have to be followed is unknown at design time, it is not possible to pre-program agents such that their behavior is norm compliant. Instead, we need a generic execution mechanism that allows agents to adapt their behavior at run-time, which is what we propose in this paper. The execution mechanism is defined on top of an abstract agent decision making mechanism. This is done by allowing the execution of actions by the agent decision making mechanism only if these are not forbidden according to norms, as well as triggering the execution of actions if this is required by norms. We specify norms using Linear Temporal Logic and define the operational semantics of the execution mechanism using techniques from executable temporal logic. We formally analyze properties of the execution mechanism, including norm compliance.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, languages and structures*; I.2.5 [**Artificial Intelligence**]: Programming Languages and Software; F.3.3 [**Logics and Meaning of Programs**]: Studies of Program Constructs; D.3.3 [**Programming Languages**]: Language Constructs and Features

Keywords

Agent Programming; Normative Systems; Formal Semantics

1. INTRODUCTION

A system of autonomous agents may exhibit undesirable or ineffective behavior if no form of regulation is imposed. An important line of research that addresses this issue is work on *normative systems*, in which norms govern the behavior of a multi-agent system (MAS). Norms describe how agents should ideally behave. The idea is that if agents enter a social context that is governed by a set of norms, they are able to recognize the norms, decide whether they want to follow them, and if they decide to do so, adapt their behavior accordingly (see, e.g., [15]). In this paper we are concerned with the last of these steps, i.e., assuming that an agent has decided to follow a set of norms, how can it generate behavior that complies with these norms.

If it is not known at design time exactly which sets of norms an agent may encounter, it is not feasible to pre-program the required behavior for each of the possible combinations of norms. Instead, we need a *generic execution mechanism* that allows agents to adapt their behavior at run-time, which is what we propose in this paper.

One way of defining such an execution mechanism is through a planning approach [16] where a plan for achieving the agent's goals is created from scratch, taking into account norms that should be satisfied. In this paper we take a different approach, and assume that an agent already has a selection of *pre-existing capabilities*, e.g., being able to pick up blocks, move around, communicate information and so forth, and a *decision making mechanism* which enables it to reason about how to use its capabilities to achieve its goals. The generic execution mechanism that allows the agent to adapt its behavior is then defined on top of these pre-existing capabilities and (abstract) agent decision making mechanism. The advantage of this approach is that it can be added on top of a variety of agent decision mechanisms.

As an example let us consider an autonomous robot intended for use in emergency search and rescue situations. We can expect this robot to be deployed in a variety of situations as part of a mixture of organisational structures; structures which may well have been assembled rapidly and may well change frequently as events progress. As such, while the robot's basic capabilities (e.g., searching buildings, removing rubble, etc.) will remain unchanged they will need to fit within different organisational protocols. For instance in some situations a robot may be expected to perform some tasks (e.g. moving rubble) only under supervision from a human; in other situations it may be trusted to complete all its tasks entirely autonomously. It is vital, therefore, that the robot can flexibly incorporate different normative rules into its reasoning.

So for instance, such a robot's basic behaviour may be to search all rooms in a building exhaustively, proceeding from room to room based upon its internal beliefs about which rooms have been sur-

[*]M. Birna van Riemsdijk was partly supported by the Dutch national programme COMMIT.

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

veyed. Now suppose this robot is placed within an organisational structure where groups of agents are assigned to each building and work together cooperatively to search it. A key to this exploration happening efficiently will be the agents *communicating* with each other to prevent individual rooms being visited by more than one robot. A typical norm for such a group might be that the robot always communicates that a room has been surveyed *before* it moves to another room. It is important to note that such a norm should not be hard-wired into the robot in advance since, the next time it is deployed, the organisational structure may be different. Also, we should reasonably expect such norms *not only* to prohibit the execution of actions but also to insist that actions not normally occurring in the agent's plans to take place (such as communicating information, requesting assistance, etc.).

Norms may exist in complicated relationships with an agent's pre-existing capabilities and decision making mechanism. Therefore treating them simply as additional beliefs, goals, or rules that the agent may adopt at run-time is insufficient, as it will be difficult (if not impossible) to do it in such a way that norm compliant behavior is generated. In order to investigate to what extent our execution mechanism can indeed guarantee norm compliant behavior, we formally specify the semantics of the execution mechanism.

To be more specific, we take the following approach in this paper. We describe an abstract agent semantics that is defined on the basis of a transition function for basic actions and a decision mechanism that chooses which of the enabled actions to execute in order to reach the agent's goals (Section 3). Then we define a language for expressing norms, which is essentially the "next-fragment" of Linear Temporal Logic [8] (Section 4). In this way we can use techniques from executable temporal logic [11] to define a normative agent semantics on *top* of the abstract agent semantics (Section 5). This semantics both *prevents* the execution of actions that would be executed according to the agent decision mechanism if this is forbidden according to norms, and *triggers* additional actions whose execution is obliged by norms. We analyze properties of these normative agent semantics in Section 6. We emphasize that in this paper we are concerned with the generation of compliant behavior. That is, we do not address reasoning about which norm to violate if no compliant behavior can be produced. Finally, we provide conclusions in Section 7, having assessed related work in Section 2.

2. RELATED WORK

Significant work has been reported at the level of the specification of normative systems and agent organizations (e.g., [9, 4, 7, 2, 14]). However, much less work has been done at the level of individual agents and how these can reason about and adapt to norms at run-time.

Meneguzzi and Luck [15] have proposed a technique for extending BDI agent languages, enabling them to adapt their behaviour at run-time in response to newly accepted norms. They do this by creating new plans to comply with obligations and suppressing the execution of existing plans that violate prohibitions. The mechanism is implemented in AgentSpeak(L) [5]. In contrast with the work reported here, no formal semantics is defined in [15] and, consequently, no properties are proved concerning the extent to which the mechanisms indeed yield compliant behavior. They also take a different approach to defining the mechanism by referring directly to BDI programming concepts such as plans. Instead, we define a novel execution mechanism on top of an abstract agent decision making mechanism. Moreover, we take a different representation of norms that is based on linear temporal logic, which allows us to use techniques from executable temporal logic to enhance the agent execution mechanism.

In [16], a framework and an implementation for norm-oriented planning is provided on top of STRIPS. It takes into consideration the operationalisation of norms during the plan generation phase, aiming for the generation of plans that both take into account the agent's utility function over actions as well as norms that should be adhered to. Thus the generation of norm compliant behavior is integrated with the agent's planning mechanism. The main difference with our work is that we define a generic execution mechanism on top of an abstract agent decision making mechanism. This allows it to be used with *any* decision making mechanism that adheres to the required form, including those defined by agent programming frameworks. Due to this different approach, the technical realization of the execution mechanism is also different from the planning approach used in [16].

In [1] the agent programming language 2APL is extended to allow reasoning about obligations and prohibitions to achieve a target state before a deadline. In that work the focus is on finding an appropriate scheduling of plans such that desired states can be achieved before their deadlines. In [1], an existing agent programming language is extended to allow reasoning about norms, rather than defining a generic execution mechanism as we do. They use a different norm language which focuses on achieving/prohibiting states before a deadline, and not a general language based on linear temporal logic to express which actions should and should not be executed. Consequently their mechanism for reasoning about norms is based on a scheduling algorithm rather than executable temporal logic as it is in our framework.

3. ABSTRACT AGENT SEMANTICS

In this section, we provide an abstract agent semantics. It defines a *transition function for basic actions* and a *decision mechanism* that chooses which of the enabled actions (according to the transition function) may be executed in a certain state. Typically the decision mechanism will be designed to assist in achieving the agent's goals. The mechanism for normative reasoning will be defined on top of this semantics.

We assume two mutually disjoint sets, namely a set of propositional atoms At with typical element p, and a set of actions Act with typical element a. We assume a language of propositional logic $\mathcal{L}_{\mathsf{At}}$ with typical element ϕ defined over At, with $\top, \bot \in \mathcal{L}_{\mathsf{At}}$ denoting the true and false sentence, respectively.

We assume a set of abstract agent states S. A partial transition function $\mathcal{T}(a, s)$ for basic actions, takes an action $a \in \mathsf{Act}$ and a state $s \in S$ and yields another state $s' \in S$ that results from executing a in s. If the action cannot be executed in s, the function is undefined.

We abstract from the external environment in which the agent operates for reasons of simplicity, assuming that the agent state accurately represents the state of the environment. Formulae $\phi \in \mathcal{L}_{\mathsf{At}}$ express properties of the agent's environment. We assume an entailment relation $s \models_{\mathcal{L}_{\mathsf{At}}} \phi$, which defines when (the agent believes that) ϕ holds in the environment.

A state s may, for example, describe the mental states of agents as in cognitive agent programming languages such as Jason/AgentSpeak [5], GOAL [13] or 2APL [6]. Then ϕ holds if, and only if, it follows from the belief base of the mental state (which is updated in each reasoning cycle with the percepts from the environment).

We define an abstract agent decision mechanism as follows. Let $\mathit{Dec}(\mathsf{Act}, S, \mathcal{T})$ be a set of transitions $s \xrightarrow{a} s'$ for which $\mathcal{T}(a, s) = s'$. This set of transitions defines which actions may be executed in which state according to the agent's decision mechanism. This is a subset of the transitions that can occur according to $\mathcal{T}$. An agent program can be used to define such a mechanism.

The abstract agent semantics for an initial state s is then defined as usual as the set of finite or infinite traces (sequences of transitions) starting in s.

DEFINITION 1. *(Abstract Agent Semantics) A trace in the agent semantics, typically denoted by t, is a finite or infinite sequence of agent states interleaved with actions $s_0, a_0, s_1, a_1, s_2, \ldots$ such that for each i (except the final state in case of a finite trace) it holds that $s_i \xrightarrow{a} s_{i+1}$ is in the set of transitions $Dec(\mathsf{Act}, S, \mathcal{T})$, and if t is a finite sequence with final state s_i then $s_i \overset{a_i}{\not\longrightarrow}$, i.e., there is no a_i, s_{i+1} such that $s_i \xrightarrow{a} s_{i+1}$.*

The semantics $S(s)$ of an agent with initial state s is the set of all traces starting in s.

4. LANGUAGE OF NORMS

Various languages for expressing norms have been proposed in the literature, and one has to choose one when defining an execution mechanism for generating norm-compliant behavior. In this paper we use a general Linear Temporal Logic (LTL) for expressing norms. The idea is that agents should exhibit behavior that complies with the temporal logic formulae.

We can express the norm from the example introduced in Section 1, namely

> "agents have to notify the other searchers that the room is surveyed before going to another room"

in LTL as follows (since we work in a propositional setting, the variables should be taken as abbreviations of their respective instantiations):

$$\Box(\text{at(Me,Room)} \wedge \text{surveyed(Room)} \rightarrow (\mathit{done}(\text{send(Searcher, surveyed(Room))}) \text{ before } \mathit{done}(\text{goTo(NewRoom)}))) \quad (1)$$

In LTL, the intuition is that

$\Box(\phi)$ means that ϕ should hold in each state of the trace, and

ϕ before ϕ' means that ϕ should hold before ϕ' holds.

The formula ϕ before ϕ' can be defined in terms of the ψ until ψ' operator - which means that ψ should hold until ψ' becomes true (where ψ' must become true at some point) - by $\neg(\neg\phi \ \mathsf{until}\ \phi')$.

It would be tempting to communicate this norm to the agents either as a goal (to inform all robots that rooms have been surveyed) or as a new rule (if you have surveyed a room, then tell the other searchers). However neither of these adequately capture the norm semantically, and both raise practical problems in typical rational agent programming languages. In particular, few agent programming languages permit complex temporal goals and so giving a robot the goal of informing other agents about some action does not guarantee this will happen in a timely fashion. It will be dependent upon the details of goal selection and discharge within the agent. Similarly we assume the agent already has a rule for exploring the building, which includes regularly leaving rooms and so if the *goTo(NewRoom)* action is invoked by this other rule it is unlikely that the rule involving communication will be selected. So we need to provide a mechanism by which a robot can reason about norms and, assuming it wishes to do so, obey them even when these norms are going to interact with its pre-existing goals and plans.

We use LTL for specifying norms since this is a very general language for specifying the desired behavior of executions. It can be used both to express what should not be done (prohibitions) as well as what should be done (obligations). We have already used LTL for specifying norms in the context of the MOISE organizational modelling language [17]. Using LTL facilitates the use of techniques from executable temporal logic [12, 11] to define a normative agent semantics. The idea of executable temporal logic is to execute a formal temporal logic specification by building a concrete model for the specification. This corresponds to our aim of generating behavior that complies with the norms specified in temporal logic. The challenge is to define how such model building can be combined with an existing agent decision mechanism into a new execution mechanism.

To facilitate executing a temporal logic specification, in [10] a normal form for temporal logic called *Separated Normal Form* (SNF) is introduced. SNF has been used as the basis for the language METATEM [11, 12], as well as for contemporary temporal resolution provers, and essentially provides a concise clausal form for temporal formulae. SNF facilitates the development of execution mechanisms for temporal logic, as it comprises the key elements required for temporal descriptions of dynamic systems: rules that constrain the current state of the system (present time rules), rules that define what the system can do next (step rules) and what it must do at some time in the future (sometime rules). Any LTL formula can be transformed into an equivalent set of SNF rules [10]. Because of its suitability for operationalization, we choose to use SNF as the basis for specifying norm-aware agents.

To provide an intuition about SNF, let us begin with an abstracted version of the formula (1) above

$$\Box(\phi \rightarrow (\mathit{done}(a_1) \ \mathsf{before}\ \mathit{done}(a_2)))$$

where $\phi \in \mathcal{L}_{\mathsf{At}}$ refers to the environment state and $\mathit{done}(a_1)$, $\mathit{done}(a_2)$ express that actions a_1 and a_2 have been executed. In order to make a clear distinction between actions and other propositional atoms, we write $\mathit{done}(a)$ rather than a (with $a \in \mathsf{Act}$). Using the translation algorithm of [10], we obtain the following SNF rules from this LTL formula:

$$\left.\begin{array}{l} \phi \Rightarrow \neg \mathit{done}(a_2) \\ \phi \Rightarrow (w \vee \mathit{done}(a_1)) \end{array}\right\} \quad \text{present time rules}$$

$$\left.\begin{array}{l} w \Rightarrow \bigcirc(\neg \mathit{done}(a_2)) \\ w \Rightarrow \bigcirc(w \vee \mathit{done}(a_1)) \end{array}\right\} \quad \text{step rules}$$

The present time rules informally specify that if the left-hand side (lhs) of the rule holds in a state, then the right-hand side (rhs) must also hold in that state. Step rules informally specify that if the lhs of the rule holds in a state, then the next state must satisfy the formula after the $\bigcirc$ operator. A third type of SNF rule is the sometime rule, which specifies that if the lhs holds, the rhs must hold at some time in the future.

In the translation process, an auxiliary atom w is introduced which can be read as 'waiting'. These atoms have a special status as they do not refer to the environment state as atoms in At, but rather are introduced for technical reasons in order to facilitate model building for the temporal formula. The way this is reflected in the normative semantics is explained in section 5. We use $\mathsf{At}_{\mathrm{aux}}$ to denote the set of auxiliary atoms, and it is important to note that a new atom is generated whenever we translate away a more complex temporal formula, such as 'before'.

In this paper we focus only on present time and step rules, i.e., on the "next-fragment" of LTL. Moreover, we focus on rules that prohibit or oblige the execution of actions, rather than achievement of a state. This means that on the rhs of rules we refer to actions (and auxiliary atoms) only. This is done for reasons of simplicity, while at the same time this fragment is expressive enough to model relevant norms as illustrated by the example. Reasoning about achievement of a state is more complex as it may involve the execution of multiple actions which are, in turn, constrained by norms. SNF rules have conjunctions of literals on their lhs and disjunctions of literals on their rhs. We define $\mathrm{lit}(\mathsf{At})$ to be the set of literals over the set of atoms At, i.e., the atoms or their negation.

We define present time rules as follows. The lhs of present time rules consists of a conjunction of literals from At and Act, and the rhs consists of a disjunction of atoms from $\mathsf{At}_{\mathrm{aux}}$. The decision to allow only a conjunction of literals from At rather than an arbitrary formula $\phi \in \mathcal{L}_{\mathsf{At}}$ on the lhs does not restrict expressiveness. A formula $\phi \in \mathcal{L}_{\mathsf{At}}$ can be translated to disjunctive normal form. Each of the disjuncts can then be translated to a separate present time rule with the disjunct on the lhs.

The choice to separate literals from different sets in the lhs and rhs does not affect the expressivity of present time rules, as, e.g., $p \wedge q \Rightarrow w \vee v$ is equivalent to $p \wedge \neg w \Rightarrow \neg q \vee v$. That is, literals can be moved freely between the lhs and rhs. We choose to separate the literals for technical convenience. By requiring literals from At and Act to be on the lhs, the applicable present time rules can be determined by evaluating the lhs, and they can be applied by considering the auxiliary atoms on the rhs (see below for details).

DEFINITION 2. *(Present Time Rules) A present time rule has the form* $\bigwedge l \Rightarrow \bigvee l'$ *where* $l \in \mathit{lit}(\mathsf{At} \cup \mathsf{Act})$ *and* $l' \in \mathsf{At}_{\mathit{aux}}$.

We define the lhs of step rules to consist of a conjunction of literals from At and Act, and atoms from $\mathsf{At}_{\mathrm{aux}}$, and the rhs to consist of a disjunction of literals from Act and atoms from $\mathsf{At}_{\mathrm{aux}}$. The translation algorithm from [10] is such that no negative auxiliary atoms will occur on the lhs or rhs. The step rules are used to specify which actions should, or should not, be executed in a state, depending on what holds in that state. Thus the lhs contains atoms that refer to properties of a state, and the rhs contains atoms referring to actions. In addition, the rhs may include auxiliary atoms to allow deferring an action, as in the example above. The rhs does not include atoms from At, as we focus on rules that prohibit or oblige the execution of actions rather than achievement of a certain state.

DEFINITION 3. *(Step Rules) A step rule has the form* $\bigwedge l \Rightarrow \bigcirc(\bigvee l')$ *where* $l \in \mathit{lit}(\mathsf{At} \cup \mathsf{Act}) \cup \mathsf{At}_{\mathit{aux}}$ *and* $l' \in \mathit{lit}(\mathsf{Act}) \cup \mathsf{At}_{\mathit{aux}}$.

5. NORMATIVE AGENT SEMANTICS

In this section, we define a normative agent semantics on top of the agent decision mechanism defined by $\mathit{Dec}(\mathsf{Act}, S, \mathcal{T})$, with the aim of generating behavior that complies with norms expressed as present time and step rules.

For this, we extend agent states in two ways, yielding normative agent states (called normative states from now on for brevity). First, we add the action that has just been executed to the state. We need this to define the applicability of present time and step rules in a state. Second, we add auxiliary atoms to the state. Normative states, typically denoted by n, thus take the form $\langle s, a, \mathit{aux}\rangle$, where $s \in S$, $a \in \mathsf{Act}$, and $\mathit{aux} \subseteq \mathsf{At}_{\mathrm{aux}}$.

We assume that normative agents can directly reference norms that they have decided to adhere to, e.g., because they are able to represent them internally. We assume that these norms are represented via a set of normative rules N, consisting of present time and step rules as specified in Definitions 2 and 3.

In order to define the normative semantics, we need to define when norms, represented as present time and step rules, are applicable in a normative state. This is the case in a normative state $n = \langle s, a, \mathit{aux}\rangle$ if a rule's lhs holds in that state, expressed as $n \models \bigwedge l$. The formula $\varphi = \bigwedge l$ holds in this extended state if, and only if, *(i)* all literals containing atoms from At in φ follow from s (according to entailment relation $\models_{\mathcal{L}_{\mathsf{At}}}$ introduced above), *(ii)* all positive literals from Act in φ are equal to a, and all negative literals containing actions from Act in φ are unequal to a, and *(iii)* all atoms from $\mathsf{At}_{\mathit{aux}}$ in φ are contained in *aux*.

DEFINITION 4. *(Applicability of Present Time and Step Rules) We say that a present time rule* $r : \bigwedge l \Rightarrow \bigvee l'$ *or step rule* $r' : \bigwedge l \Rightarrow \bigcirc(\bigvee l')$ *is applicable in a normative state* n *iff* $n \models \bigwedge l$. *We denote this as* $\mathit{applicable}(r, n)$.

If norms are applicable, they may affect the choice of action of the agent. The semantics needs to be defined such that transitions are generated that comply with the (applicable) norms, i.e., that make the rhs of the applicable norms true (from one state to the next for step rules, and for each state for present time rules).

The basic idea for generating such a transition is taken from the METATEM [3, 11] execution semantics. Since the rhs of present time and step rules consist of a disjunction of literals, these can be satisfied by making one of these literals true. For example, in order to satisfy $(w_1 \vee w_2)$ and $(w_3 \vee w_4)$, one needs to satisfy $(w_1 \wedge w_3)$ or $(w_1 \wedge w_4)$ or $(w_2 \wedge w_3)$ or $(w_2 \wedge w_4)$. Each of these conjunctions of literals represents a *choice* for the agent that would allow it to generate a transition that complies with norms. Thus we call this set of conjunctions the *normative choices*.

Below we present a formalization of the set of normative choices for a set of norms N and a transition from normative state n to n', denoted as $\mathit{NChoices}(N, n, n')$. We will use the formalisation to generate the set of auxilliary literals that may appear in n'. Intuitively $\mathit{NChoices}(N, n, n')$ captures all the auxilliary and action literals that can appear in n' as dictated by the present time rules applied to n' and the step rules applied to n. At this point we make no assumption that n itself complies with our norms; we are interested only in the compliance of the 'new' state n'. The formalization takes the rhs of step rules applicable in n and present time rules that are applicable in n'. This set of disjunctions is transformed to a set of conjunctions of literals as sketched above. From the resulting set of conjunctions, we take only those that are satisfiable in n'. For example, if a conjunct expresses that action a should be executed and the normative state n' contains a different action, the conjunct is not satisfiable in n'.

The set of normative choices is thus defined with respect to a given transition (n and n' are taken as parameters), i.e., we use the set of normative choices to check whether a given transition is compatible with one of the possible normative choices. The only exception is the auxiliary atoms in n'. In the definition, all elements of n and n' are used, *except* the auxiliary atoms of n': only the check for applicability of present time rules uses n', and present time rules do not have auxiliary atoms on their lhs. The definition of normative choices is thus independent of the auxiliary atoms in n'. The generation of appropriate auxiliary atoms for n' is taken care of in the transition rules that follow.

DEFINITION 5. *(Normative Choices) Let* N *be a set of normative rules and let* n, n' *be normative states. Let* $\mathit{PTR}(N)$ *and* $\mathit{SR}(N)$ *be the sets of present time and step rules of* N*, respectively. We use* $\mathit{rhs}(r)$ *to denote the disjunction of literals in the rhs of rule* r*. We then define* $\mathit{NChoices}(N, n, n')$ *as*

$$\mathit{sat}(\mathit{CNF2DNF}(\{\mathit{rhs}(r) \mid (r \in \mathit{SR}(N), \mathit{applicable(r,n)}) \text{ or } (r \in \mathit{PTR}(N), \mathit{applicable(r,n')})\}), n')$$

where CNF2DNF transforms a set of disjunctions to an equivalent set of conjunctions (where the set $\{\top\}$ is returned in case the set of disjunctions is empty), and $sat(C, n')$ deletes those conjuncts from C that are not satisfiable in n', i.e., those that contain a positive action literal different from the action of n', a negative action literal containing the action of n', and those that are inconsistent.

It is important to note that if $\mathit{NChoices}(N, n, n') = \emptyset$ (which can only occur if *sat* removes all conjunctions), there are no norm compliant ways of making the transition from n to n'. Moreover, if there are no applicable present time or step rules, it means that the agent is not constrained by norms in this transition. We return $\{\top\}$ in this case (note that we should not return $\emptyset$, as this would mean precisely the opposite, namely that there are no norm compliant ways to make the transition). Definition 5 is such that no normative choice in $\mathit{NChoices}(N, n, n')$ will contain more than one positive action literal, as this would make it unsatisfiable in n'.

We now define two transition rules that formalize the normative agent semantics. The first rule is for executing an action that can be taken according to the agent decision mechanism. This action selection is *constrained* by norms, as only actions may be executed that are not forbidden according to norms.

The second transition rule is for *triggering action execution* according to *step rules*. This concerns actions that the agent may not have taken according to its decision mechanism, but which it is obligated to take by its norms as expressed by step rules. This does not necessarily mean that the action *has* to be executed at that moment. In the example norm from Section 4, if w holds in a normative state then both step rules are applicable. This would allow the agent to do any action other than a_2, e.g., some action a_3. The semantics should however also allow the agent to select action a_1 in this case, as it will have to be executed at some point (that is, if the agent eventually wants to execute a_2). The second transition rule does precisely this: allowing the execution of actions that are (eventually) required by step rules, but that would not (necessarily) be executed according to the agent decision mechanism.

We formally define the first transition rule, action semantics constrained by norms, as follows. A transition can be generated from a normative state $n = \langle s, a, \mathit{aux}\rangle$ with action a' if there is a transition for action a' from s to s' in the agent decision mechanism, and there is a normative choice that is compatible with this action. We say that a normative choice $c \in \mathit{NChoices}(N, n, n')$ is compatible with action a', denoted as $\mathit{compatible}(c, a')$, if and only if c contains no positive action literals, or c contains a single positive action literal equal to a'. The next normative state is of the form $\langle s', a', \mathit{aux}'\rangle$, where $\mathit{aux}' = aux(c)$, which denotes the atoms l from conjunct c such that $l \in \mathsf{At}_{\mathsf{aux}}$. In this way the auxiliary atoms for the next normative state are generated to comply with norms.[1]

DEFINITION 6. *(Action Semantics Constrained by Norms) Let $n = \langle s, a, \mathit{aux}\rangle$ and $n' = \langle s', a', \emptyset\rangle$. Then the transition rule for execution of actions constrained by norms, is defined as follows.*

$$\frac{s \xrightarrow{a'} s' \in \mathit{Dec}(\mathsf{Act}, S, \mathcal{T}) \qquad c \in \mathit{NChoices}(N, n, n') \quad \mathit{compatible}(c, a') \quad \mathit{aux}(c) = \mathit{aux}'}{n \xrightarrow{a'} \langle s', a', \mathit{aux}'\rangle}$$

Note that $\mathit{compatible}(\top, a)$ for all actions a as $\top$ does not contain positive action literals.

[1]Note that we generate the set of choices, $\mathit{NChoices}(N, n, n')$, using a normative state n' in which the set of auxilliary atoms is empty, this guarantees the maximal set of choices for norms N, state n, and action a'. We define the semantics in this way to bootstrap generation of the next normative state, who's auxiliary atoms are derived from one of the normative choices.

The second transition rule, step rule semantics for triggering action execution, is defined as follows. A transition can be generated from a normative state $n = \langle s, a, \mathit{aux}\rangle$ with action a' if there is a step rule in which a' occurs positively in its rhs and which is applicable in n, and a' is enabled in n according to the transition function for actions, and there is a normative choice that is compatible with a'. Then, similarly to the first transition rule, the next normative state is of the form $\langle s', a', \mathit{aux}'\rangle$, where $\mathit{aux}' = aux(c)$.

DEFINITION 7. *(Step Rule Semantics for Triggering Action Execution) Let $n = \langle s, a, \mathit{aux}\rangle$ and $n' = \langle s', a', \emptyset\rangle$. Let $r : \bigwedge l \Rightarrow \bigcirc(\bigvee l')$ be a step rule. In order to ensure satisfaction of this step rule over a transition, the agent can execute one of the actions $a_0, \ldots, a_n$ that occur positively in $\bigvee l'$, denoted by $a \in \bigvee l'$. This is modelled in the following transition rule.*

$$\frac{n \models \bigwedge l \quad a' \in \bigvee l' \quad \mathcal{T}(a', s) = s' \qquad c \in \mathit{NChoices}(N, n, n') \quad \mathit{compatible}(c, a') \quad \mathit{aux}(c) = \mathit{aux}'}{n \xrightarrow{a'} \langle s', a', \mathit{aux}'\rangle}$$

The transition relation $\xrightarrow{a}$ is the smallest relation induced by the two transition rules defined above. The normative agent semantics for an initial state s and set of norms N is then the set of finite or infinite traces (sequences of transitions) generated by the transition rules above and starting in a normative state that has s as the first element. These initial states have to comply with present time rules in order to ensure that the initial state is norm compliant.

DEFINITION 8. *(Normative Agent Semantics) A trace in the normative transition system (under set of norms N), typically denoted by t and referred to as a normative trace, is a finite or infinite sequence of normative states interleaved with actions $n_0, a_0, n_1, a_1, n_2, \ldots$ such that for each i (except the final state in case of a finite trace) the transition $n_i \xrightarrow{a_i} n_{i+1}$ can be derived using a transition rule of Definition 6 or 7. If t is a finite sequence with final state n_i then $n_i \not\xrightarrow{a_i}$.*

Let $\mathit{NChoices}(N, n)$ be a variant of $\mathit{NChoices}(N, n, n')$ where only present time rules are considered and their applicability is with respect to n. Let $n = \langle s, \epsilon, \emptyset\rangle$ and $\mathit{NChoices}(N, n) = C$. The normative semantics $S^N(s)$ of an agent starting in state s is undefined if $C = \emptyset$. Otherwise, we define the set of initial normative states of the agent as $\mathit{Init}(s) = \{\langle s, \epsilon, aux(c)\rangle \mid c \in C\}$ (where ϵ denotes the empty action). Then $S^N(s)$ is the set of all traces under set of norms N starting in a normative state $n_0 \in \mathit{Init}(s)$.

COROLLARY 1. *If N is empty (or if none of the norms are ever applicable), then the set of traces generated by the transition rules are equivalent to those of the original agent program modulo extension of state to normative states.*

6. PROPERTIES

In this section we investigate properties of the normative semantics. We start with a weak definition of what it is for a trace to satisfy a set of norms and show that this holds true of the execution mechanism we have defined. We then move on to examine stronger definitions that capture the idea that the trace has not been halted because further action is forbidden by the norms.

6.1 Weak Norm Compliance

In order to investigate norm compliance, we first need to make precise when a state and transition satisfy present time and step rules, respectively. Definition 9 defines satisfaction of present time rules in a normative state. A present time rule is satisfied in a state if it holds that if the lhs holds in that state, then the rhs holds in that state. We lift this definition to sets of present time rules as expected.

DEFINITION 9. *(Satisfaction of Present Time Rules) Let n be a normative state and let N be a set of norms. We define that a present time rule $r : \bigwedge l \Rightarrow \bigvee l'$ is satisfied in n iff it holds that if $n \models \bigwedge l$ then $n \models \bigvee l'$. We define that SatPresent$^N(n)$ holds iff all present time rules in N are satisfied in n.*

Definition 10 defines satisfaction of step rules over a pair of normative states. A step rule is satisfied over a pair of normative states if it holds that if the lhs holds in the first state of the pair, then the rhs holds in the second state. Again the definition is lifted to sets of step rules. Note that this definition does not specify that there is a transition from n to n' in the transition relation. This is enforced in definitions below.

DEFINITION 10. *(Satisfaction of Step Rules) Let n, n' be normative states and let N be a set of norms. We define that a step rule $r : \bigwedge l \Rightarrow \bigcirc(\bigvee l')$ is satisfied over (n, n') iff it holds that if $n \models \bigwedge l$ then $n' \models \bigvee l'$. We define that SatStep$^N(n, n')$ holds iff all present time rules in S are satisfied over (n, n').*

We identify two types of norm compliance: weak and strong compliance. We define that a trace weakly complies with a set of norms if all states and transitions on that trace satisfy the norms.

DEFINITION 11. *(Weak Norm Compliance) Let s be an agent state and N a set of norms. Then a normative trace $t \in S^N(s)$ weakly complies with N iff for each state n on t it holds that SatPresent$^N(n)$, and for each two consecutive states n_i, n_{i+1} on t it holds that SatStep$^N(n_i, n_{i+1})$.*

We prove that our semantics is weakly compliant with norms.

THEOREM 1. *Let s be an agent state and N a set of norms. Then for all normative traces $t \in S^N(s)$ it holds that t weakly complies with N.*

Proof: By induction over the length of traces. Initial states of t satisfy present time rules by Definition 8. To prove: if n_i is a state on t and *SatPresent*$^N(n_i)$, then it holds that *SatPresent*$^N(n_{i+1})$ and *SatStep*$^N(n_i, n_{i+1})$. This follows from the fact that the transition from n_i to n_{i+1} has to be generated by a transition rule from Definition 6 or 7. These can only be applied if the applicable norms are respected (in n_{i+1} for present time rules and over (n_i, n_{i+1}) for step rules) or if there are no applicable norms (which yields $C = \{\top\}$), as specified through *NChoices*(N, n, n') (Definition 5). □

Our semantics is weakly norm compliant as no next transition will be generated if all potential next transitions would violate present time or step rules. This has the effect that if the agent runs into a situation where it cannot do anything without violating norms, it will stop. It may seem as if this property does not completely capture the desired behavior of the normative semantics, as a normative semantics that does not generate any transitions would also satisfy it. However, in combination with Corollary 1 - which says that if there are no norms, the normative semantics will generate the original agent traces - we can conclude that the semantics not only generates traces that satisfy norms, but also keeps the original agent semantics intact if possible.

One way to address these problems is to add reasoning mechanisms that allow agents to *choose which norms to violate*, if they cannot generate compliant behavior for all of them. Adding such reasoning mechanisms is left for future work. Another way to address this is to investigate how one can *prevent* agents from running into a situation where all they can do is violate norms. In this paper we characterize these situations and identify different settings (as properties of a set of traces) in which an agent will, may, or will not get into these situations. It is left for future work to investigate which agent and norm properties give rise to occurrence of these different settings.

6.2 Strong Norm Compliance

In this section we examine situations where the agent can not proceed without violating norms where the conflicts arise from interactions between the norms, i.e., situations where any action that could in principle be executed would violate a norm. In section 6.3 we extend this to situations where we consider only actions that would be selected by the decision mechanism.

The characterization of situations in which the agent would stop is done by identifying different types of conflict: practical and normative conflict. These definitions apply to an action a' that is a *normative option* in n, meaning that it occurs as a positive action literal on the rhs of an applicable step rule in that state. The conflicts arise because this action cannot be executed either because it is practically impossible (because $\mathcal{T}(a', s)$ is undefined), or it conflicts with other norms (they either specify actions which have to be executed at the same time, or some norm forbids the execution of a' in n). Intuitively, practical and normative conflicts should be avoided if possible, in particular in cases where the transition rule of Definition 6 is not applicable. In these cases, the agent gets stuck because it cannot satisfy norms.

DEFINITION 12. *(Normative Option) Let $n = \langle s, a, aux\rangle$ be a normative state and let N be a set of norms. We define that action a' is a* normative option *in n with respect to N, denoted as option$^N(a', n)$, iff there is a step rule $r : \bigwedge l \Rightarrow \bigcirc(\bigvee l') \in N$ and applicable(r, n) and done$(a') \in \bigvee l'$.*

Definition 13 now describes a situation where an agent cannot proceed in a normative fashion because its step rules require some action to be taken which is not enabled according to the transition function for actions.

DEFINITION 13. *(Practical Conflict) We define a* practical conflict *of action a' in n with respect to set of norms N, denoted as pracConflict$^N(a', n)$ as follows: pracConflict$^N(a', n)$ holds iff option$^N(a', n)$ and $\mathcal{T}(a', s)$ is undefined.*

Definition 14 describes situations in which an agent's norms interact in such a way that *either* it is forced to take two actions at once *or* it is forced to take an action by one norm that is forbidden by another.

DEFINITION 14. *(Normative Conflict) Let $n = \langle s, a, aux\rangle$, $\mathcal{T}(a', s) = s'$ and $n' = \langle s', a', \emptyset\rangle$. We define the set of potential normative choices, PNChoices(N, n, n'), as NChoices(N, n, n') without removing conjuncts that are unsatisfiable in n' (except those that are unsatisfiable because of inconsistencies between auxiliary atoms; this can occur if the norms themselves are inconsistent with each other). We define a* normative conflict *of action a' in n with respect to set of norms N, denoted as normConflict$^N(a', n)$ as follows: normConflict$^N(a', n)$ holds iff option$^N(a', n)$ and for all $c \in$ PNChoices(N, n, n') it holds that if done$(a') \in c$, then there is another positive literal done$(b) \in c$, or another negative literal $\neg$done$(a') \in c$.*

We now show that these two types of conflict *completely characterize* situations in which an agent cannot make a transition when a normative option is available, i.e., the only situations in which an agent cannot make a transition when a normative option is available is when that option is either in practical or normative conflict with the set of norms.

THEOREM 2. *Let $n = \langle s, a, aux\rangle$ be a normative state and let N be a set of norms such that SatPresent$^N(n)$. If $n \overset{a}{\not\longrightarrow}$ then for all a' such that option$^N(a', n)$, either normConflict$^N(a', n)$ or pracConflict$^N(a', n)$.*

Proof: (By contradiction). Assume there exists some a' such that *option*$^N(a', n)$ and $n \overset{a}{\not\longrightarrow}$ but neither *normConflict*$^N(a', n)$ nor *pracConflict*$^N(a', n)$. We seek to prove the existence of a normative state $\langle s', a', aux'\rangle$ allowed by Definition 7 (Step rule Semantics for Triggering Action Execution). Most of the preconditions of this rule follow trivially from our assumptions, leaving us only to establish that given $n' = \langle s', a', \emptyset\rangle$ there exists some $c \in$ *NChoices*(N, n, n') such that $compatible(c, a')$. Because $\neg$*normConflict*$^N(a', n)$ we know there exists $c \in$ *PNChoices*(N, n, n') such that $\neg done(a') \notin c$ and $\forall b \neq a'.\ done(b) \notin c$ so $compatible(c, a')$. Therefore $c \in$ *NChoices*(N, n, n'). □

Based on these definitions of conflict, we now define a strong form of norm compliance. We say that a trace is strongly norm compliant if it avoids situations of practical or normative conflict. This identifies traces in which if the norms could, potentially, trigger an action that action may be taken.

DEFINITION 15. *(Strong Norm Compliance) Let s be an agent state. Then a trace $t \in S^N(s)$ strongly complies with the set of norms N iff t is infinite, or t is finite and for the final state n_f of t it holds that there is no action a' such that pracConflict$^N(a', n_f)$ or normConflict$^N(a', n_f)$.*

Our semantics does not in general adhere to strong norm compliance, i.e., it is easy to construct an example where traces are generated that do not strongly satisfy norms. Thus what we need to do in order to get a better understanding of strong norm compliance is identify different settings (as properties of a set of traces) in which an agent will, may, or will not get into situations of conflict. We distinguish the following interesting cases:

1. A set of norms N is conflicting: for all agent decision functions *Dec*$(\mathsf{Act}, S, \mathcal{T})$ and initial agent states $s \in S$, it holds that none of the traces in $S^N(s)$ strongly complies with norms.
2. A set of norms N conflicts with agent decision function *Dec*$(\mathsf{Act}, S, \mathcal{T})$: for all agent states $s \in S$, it holds that none of the traces in $S^N(s)$ strongly complies with norms.
3. A set of norms N is strongly satisfiable with agent decision function *Dec*$(\mathsf{Act}, S, \mathcal{T})$ and initial state s: there is a trace in $S^N(s)$ that strongly complies with norms.
4. A set of norms N is strongly satisfied with agent decision function *Dec*$(\mathsf{Act}, S, \mathcal{T})$: for all agent states $s \in S$, it holds that all traces in $S^N(s)$ strongly comply with norms.
5. A set of norms N is strongly satisfied: for all agent decision functions *Dec*$(\mathsf{Act}, S, \mathcal{T})$ (or agent decision functions that satisfy certain properties in relation to the norms) and all initial agent states $s \in S$, it holds that all traces in $S^N(s)$ strongly comply with norms.

A conflicting set of norms should be avoided. If a set of norms conflicts with an agent decision function, the agent should try to avoid situations where it is exposed to these norms. The challenge then is to determine whether this is the case before it starts execution under these norms. If a set of norms is strongly satisfiable with an agent decision function and initial state, computational reasoning mechanisms should be developed that allow the agent to choose a trace which strongly complies with norms. If a set of norms is strongly satisfied with an agent decision function, there is no need to develop computational reasoning mechanisms as all traces will strongly comply with norms. The challenge is then to determine whether this is the case before the agent starts execution, to avoid the overhead of reasoning about which trace to choose. Concerning sets of norms that are strongly satisfied, the interesting challenge is to find conditions (restrictions on norms) for which this is the case. Then if norms adhere to these conditions, every agent can function in this set of norms. Identifying such cases is future work.

6.3 Normative Decision Compliance

While strong norm satisfaction identifies situations in which norms are unable to trigger the actions they require, we are also interested in investigating situations where the norms interfere with the agent's execution. Clearly the purpose of norms is to modify the behaviour of an agent so we need to identify situations in which this modification is detrimental.

We define *prohibition conflict* to capture a situation where the agent decision mechanism could choose to make a transition were it not prevented by the norms.

DEFINITION 16. *(Prohibition Conflict) Let $n = \langle s, a, aux\rangle$ be a normative state and let N be a set of norms. If $s \xrightarrow{a'} s' \in$ Dec$(\mathsf{Act}, S, \mathcal{T})$ and $n' = \langle s', a', \emptyset\rangle$ we say that a' is in* prohibition conflict *proConflict$^N(a', n)$ if for all $c \in$ PNChoices(N, n, n') either $\neg done(a') \in c$, or there is a positive literal $done(b) \in c$ such that $b \neq a'$.*

There are whole classes of norms for which prohibition conflicts are desirable. The problems arise in situations in which the agent can take no action at all because of such conflicts. These would appear in finite normative traces from which the final state can make no transition because of a prohibition conflict.

DEFINITION 17. *(Normative Decision Compliance) Let s be an agent state. Then a trace $t \in S^N(s)$ is said to* preserve *the agent decision mechanism, Dec$(\mathsf{Act}, S, \mathcal{T})$ iff t is infinite, or t is finite and for the final state n_f of t it holds that there is no action a' such that pracConflict$^N(a', n_f)$, normConflict$^N(a', n_f)$ or proConflict$^N(a', n)$.*

Note that a trace which preserves a decision mechanism also strongly complies with the norms.

As above we identify a number of interesting cases.

1. A set of norms N potentially preserves decision mechanism *Dec*$(\mathsf{Act}, S, \mathcal{T})$ for initial state s: there is a trace in $S^N(s)$ that preserves *Dec*$(\mathsf{Act}, S, \mathcal{T})$.
2. A set of norms N preserves decision mechanism *Dec*$(\mathsf{Act}, S, \mathcal{T})$: for all agent states $s \in S$, it holds that all traces in $S^N(s)$ preserve *Dec*$(\mathsf{Act}, S, \mathcal{T})$.

The existence of a trace that preserves a decision mechanism does not guarantee that an agent can achieve its goals – for instance the norms may have forced the agent to take actions that place it in a situation where no action can be chosen by the decision mechanism even though there is no conflict with the norms. In this situation the agent can do nothing further even though its goals have not been achieved. However a normative decision compliant trace is one in which the agent has not ended up in a situation where the only actions it would choose to take are ones that would violate a norm.

It is clearly desirable for an agent to identify strategies that will lead to decision mechanism preserving traces. In particular it will be useful for an agent to have strategies for those occasions when a transition is possible under both the transition rule of Definition 6 and the rule of Definition 7. Two possible strategies are a *lazy* strategy which always prefers transitions under rule 6 and delays all norm-triggered actions until the last moment and a *greedy* strategy which always prefers to take a norm-triggered action to one suggested by the decision function.

It is possible, with both strategies, to construct examples in which there are decision mechanism preserving traces which cannot be found by the strategy. In the first case delaying norm-triggered actions can lead to situations in which it becomes "too late" and a normative conflict arises in which the agent must take two actions at once or a practical conflict arises in which the agent is no longer able to take the action required by the norm. In the second case an over-eager strategy can lead to prohibition conflicts where the taking of a normative action leads to a state in which the norms prevent an action being taken which could have been taken had the norm-triggered action been delayed. Future research will have to identify situations in which specific strategies can be shown to lead to decision mechanism preserving traces.

7. CONCLUSION AND FUTURE WORK

In ensuring that agent programs can follow norms in a range of different scenarios we cannot pre-compute all such possible normative behaviours at design time. Consequently, a promising route is to have a separate, normative layer, combined with an agent reasoning mechanism that can take norms into account at run-time. We have proposed an approach in which a fragment of LTL is used to express norms, techniques from executable temporal logic define the operational semantics of the norm-aware execution mechanism. This mechanism abstracts from the agent decision function. Thus as agents move between different contexts the norms will change, yet the agent decision mechanism remains the same.

As all LTL formulae can be transformed into an equivalent set of SNF rules, we have taken a subset of these rules just concerning the present and next steps of the execution. Clearly this represents only a fragment of LTL, but it allows us to describe simple norms, such as "do A before doing B", in a straightforward manner. The SNF rules corresponding to the normative behaviour then provide a form of "normative control" on agent execution.

Of course, since we only consider a fragment of executable temporal logic, we cannot say that this mechanism is complete for any temporal constraint we might provide. However, for norms of the form mentioned above, this mechanism generates behavior that (weakly) complies with norms.

We consider the following questions, among others, for future work. It will be interesting to investigate the effect of complying with norms on goal achievement, and investigate how one can guarantee that such goals are achieved if the agent adapts its behavior to comply with norms (if the goals were achieved in the original agent semantics). Also we will address the question of identifying when a set of norms is strongly satisfied. This would allow individual agents and potentially the designers of organisations to decide when it is appropriate for some agent to adopt a set of norms, or whether it is appropriate to impose some set of norms upon all the agents in an organisation. Similarly given a set of norms, we are interested in identifying strategies that will find traces that are strongly norm compliant, or preserve the decision mechanism. Finally, it will be interesting to extend the framework with a mechanism for reasoning about norm violation.

8. REFERENCES

[1] N. Alechina, M. Dastani, and B. Logan. Programming Norm-aware Agents. In *Proc. of AAMAS'12*, pages 1057–1064. IFAAMAS, 2012.

[2] L. Astefanoaei, M. Dastani, J.-J. Ch. Meyer, and F.S. de Boer. On the Semantics and Verification of Normative Multi-Agent Systems. *J. UCS*, 15(13):2629–2652, 2009.

[3] H. Barringer, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds, editors. *The Imperative Future: Principles of Executable Temporal Logics*. Research Studies Press, 1996.

[4] G. Boella and L. van der Torre. Regulative and constitutive norms in normative multiagent systems. In *Proc. of KR'04*, pages 255–265. AAAI Press, 2004.

[5] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-agent Systems in AgentSpeak using Jason*. Wiley, 2007.

[6] M. Dastani. 2APL: a practical agent programming language. *JAAMAS*, 16(3):214–248, 2008.

[7] V. Dignum. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. PhD thesis, 2004.

[8] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 996–1072. Elsevier, Amsterdam, 1990.

[9] M. Esteva, J. Padget, and C. Sierra. Formalizing a language for institutions and norms. In J.-J. Ch. Meyer and M. Tambe, editors, *Proc. 8th International Workshop on Intelligent Agents (ATAL)*, volume 2333 of *LNCS*, pages 348–366. Springer, 2002.

[10] M. Fisher. A normal form for temporal logics and its applications in theorem-proving and execution. *Journal of Logic and Computation*, 7(4):429–456, 1997.

[11] M. Fisher. Agent Deliberation in an Executable Temporal Framework. *Journal of Applied Logic*, 9(4):223–238, 2011.

[12] M. Fisher and A. Hepple. Executing logical agent specifications. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 1–27. Springer, Berlin, 2009.

[13] K. V. Hindriks. Programming rational agents in GOAL. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*. Springer, Berlin, 2009.

[14] J. F. Hübner, O. Boissier, and R. H. Bordini. A Normative Programming Language for Multi-agent Organisations. *Ann. Math. Artif. Intell.*, 62(1-2):27–53, 2011.

[15] F. Meneguzzi and M. Luck. Norm-based behaviour modification in BDI agents. In *Proc. of AAMAS'09*, pages 177–184, Budapest, 2009.

[16] S. Panagiotidi and J. Vázquez-Salceda. Norm-aware planning: Semantics and implementation. In *Proc. of WI-IAT'11*, pages 33–36. IEEE, 2011.

[17] M. B. van Riemsdijk, K. V. Hindriks, C. M. Jonker, and M. Sierhuis. Formalizing organizational constraints: A semantic approach. In *Proc. of AAMAS'10*, pages 823–830. IFAAMAS, 2010.