Norm-based behaviour modification in BDI agents

Felipe Meneguzzi
King's College London
Department of Computer Science
London, United Kingdom
felipe.meneguzzi@kcl.ac.uk

Michael Luck
King's College London
Department of Computer Science
London, United Kingdom
michael.luck@kcl.ac.uk

ABSTRACT

While there has been much work on developing frameworks and models of norms and normative systems, consideration of the impact of norms on the practical reasoning of agents has attracted less attention. The problem is that traditional agent architectures and their associated languages provide no mechanism to adapt an agent at runtime to norms constraining their behaviour. This is important because if BDI-type agents are to operate in open environments, they need to adapt to changes in the norms that regulate such environments. In response, in this paper we provide a technique to extend BDI agent languages, by enabling them to enact behaviour modification at runtime in response to newly accepted norms. Our solution consists of creating new plans to comply with obligations and suppressing the execution of existing plans that violate prohibitions. We demonstrate the viability of our approach through an implementation of our solution in the AgentSpeak(L) language.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms, Design, Languages

Keywords

Norms, BDI

1. INTRODUCTION

Systems composed of heterogeneous autonomous agents require some form of societal control to ensure a desirable social order in which agents work together effectively. For many, *norms* are the mechanism of choice to address this concern in multiagent societies and ensure order and predictability [1]. Such norms define standards of behaviour that are acceptable in a society, indicating desirable behaviours that should be carried out, as well as undesirable behaviours that should be avoided. Normative systems thus rely on a representation of obligations, prohibitions and permissions that ensure that complying agents act within some predefined bounds. Although deontic concepts have received much attention from philosophy [12, 5], and more recently computer science [7, 8], we must provide a simple definition for the concepts we use in this paper. Therefore, we take an obligation to be a *positive* constraint on an agent, indicating that it *must act* to accomplish something in the world [10]; permissions to be overriders of obligations, undercutting them and freeing an agent from being bound to a particular constraint;and prohibitions to be negative constraints on an agent, indicating that it must *refrain* from acting in a particular way [10].

The specification and maintenance of normative systems has been the focus of recent research in agents, for example in the context of electronic institutions [1]. However, these efforts have been largely at the macro level, such as managing global sets of norms in an environment [14] and monitoring an agent's actions within a society for compliance [4], among others. By contrast, little work has been done on dealing with the desired *effects* of norms at the level of an individual agent, and traditional agent architectures are generally lacking in mechanisms to adapt agents to comply with newly perceived norms at runtime.

In this paper we develop a solution that enables agents to process norms and modify their behaviour so as to comply with these norms, should they choose to do so. In order to accomplish this, we extend a BDI language allowing it to modify plans at runtime in reaction to newly accepted norms. Our main contribution is the introduction of plan manipulation strategies to enable reasoning about norms and to ensure compliance with newly accepted norms.

The paper is structured as follows: in Section 2 we outline our understanding of norms in agent languages; in Section 3 we describe our behaviour modification algorithms; in Section 4 we describe an AgentSpeak(L)-based implementation of our solution using a meta-reasoning toolkit; and in Section 5 we summarise relevant related work and conclude the paper.

2. NORMS IN AGENT LANGUAGES

2.1 Norm Representation

In order to add normative processing to an agent language, it is necessary to provide some sort of representation for norms. Norms are often situated within the context of an electronic institution, making norms part of the environment [14]. Here, however, we are not concerned with the issues of electronic institutions or the modelling of a normative environment, but with the aspects that arise after norms have been perceived by an agent. We are concerned with the operational reasoning of agents, so our focus is on the consequences of norm compliance on an agent reasoning process. It is important to note that there are two major perspectives regarding normative systems: one with norms that, *by design*, cannot be violated, which is the perspective taken by electronic institutions [1]; and another with norms that *can* be violated, potentially entailing penalties for the violator [5]. The latter perspective has been gaining increasing acceptance within computer science [10, 4], since

Cite as: Norm-Based Behaviour Modification in Bdi Agents, Felipe Meneguzzi, Michael Luck, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 177–184

Norm	Meaning
$obligation(p)$	add a goal to achieve state p, from $Activation$ to $Expiration$.
$obligation(a)$	add a new plan with a $Activation$ triggering event, and action a in its body.
$prohibition(p)$	prevent adoption of plans that bring about state p.
$prohibition(a)$	prevent adoption of plans that execute action a.

Table 1: Summary of norms and their meaning.

it considers that autonomous agents must always have choice over their behaviour. This paper focuses on autonomous agents and how to achieve flexible behaviour, thus we take precisely this approach. From an individual agent's perspective, the main effect of norms on its reasoning is to determine which behaviours *must* be carried out, and which behaviours *must not* be carried out or else some form of punishment ensues. Of course, other types of norms prescribing preferences or more complex stipulations are possible, but from a practical perspective, and using a closed world assumption where everything not prohibited is permitted, the possibility of non-binding recommendations can be ignored, leaving an agent's own reasoning process to determine the best courses of action. This means that we ignore permissions, since they just recommend behaviours rather than *require* behaviours to be changed. Although we could include permissions as norm-modification operators in the sense that a permission may invalidate an obligation or prohibition, the resulting system would have effectively the same functionality in terms of behaviour modification yet would complicate the discussion in this paper.

2.2 Norms and Goal Types

Norms may refer to either: declarative world states, in which an agent must try to achieve or refrain from achieving certain world states; or actions, in which an agent must try to execute or refrain from executing a particular action [6], as is summarised in Table 1.

Norms must have a well defined validity period; that is, a specification of when a certain norm is in force and when it ceases to be in force. This validity period is crucial for the enforcement of norms, particularly in the case of obligations, as without a deadline, an obliged party is not compelled to fulfill its obligation at any particular time. For example, if a consumer is obliged to pay his or her electricity bill, it is not possible to detect a violation without knowing when this payment is due. Conversely, for prohibitions, certain industrial research positions require a researcher to sign a form of non-disclosure agreement that includes provisions forbidding the researcher to engage in any competing research for a set period of time, usually one year after the termination of the initial contract. Therefore, norm encodings normally include a representation of the activation and expiration of each norm, indicating when the deontic modality referred to in the norm (*e.g.* obligation and prohibition) should be complied with.

Now, since most agent languages, such as AgentSpeak(L), 3APL, and others [2] use first-order logic to represent beliefs, it is appropriate to adopt a similar type of norm representation. Such a logic-based representation of norms allows for a more straightforward use of an agent language in detecting when norms are being complied with. Moreover, we leverage representational concepts from the formalisation of Oren *et al.*[11], which includes notions of activation and expiration of a norm, delimiting their validity through time, an important aspect of norms in a dynamic environment. We therefore adopt a representation for norms in our system as follows:

$$norm(Activation, Expiration, Norm)$$

where $Activation$ is the *activation condition* for the norm to become active, $Expiration$ is the *expiration condition* to deactivate the norm, and $Norm$ is the norm itself. For example, if an agent is obliged to drive on the left when it is in Britain, but not when it leaves, the norm denoting this obligation is represented as `norm(in(britain), notIn(britain), obligation(driveOn(left)))`. In this context, we focus on using norms to determine whether plans and actions may be executed, and on the introduction of new triggering events linked to plans to satisfy new obligations.

It is important to note that we are not concerned, at this point, with handling more complex norm representation schemes in the way that a complete deontic logic does. Rather, we reduce norm representation to these two outcomes of prohibition and obligation to facilitate the creation of concrete agent behaviours aimed at complying with a set of norms. Ultimately, however, norms created with a more complex representation language must be reduced to these two outcomes in order to enable meaningful modifications to an agent's behaviour to be inferred.

There are many possible interpretations for such activation and expiration conditions, so it is important to specify what each condition means for an agent. We consider the activation condition to be a logical formula that, once entailed by an agent's beliefs, results in the norm being applicable to the agent. Conversely, we consider the expiration condition to be a logical formula that, once entailed by an agent's beliefs, results in the norm ceasing to be applicable to the agent. In what follows, we assume these conditions under a monotonic logic framework, so that when a condition becomes true, it will remain true in the future. We make this assumption now in order to simplify the description of our behaviour modification technique, so that norms can be activated and expire only once through their activation and expiration conditions. In this way, once a norm has been accepted and acted upon, there is no need for an agent to keep track of which norms have been accepted in the past, since monotonicity ensures their activation does not occur multiple times. We can later drop this assumption, but it is useful to keep it for ease of explanation, eliminating the need for more elaborate bookkeeping in our algorithms.

2.3 Norm Perception

In relation to *accepting* a norm, we consider its life cycle relative to an agent to start with the norm being issued in an environment (or society), and perceived by an agent. When an agent perceives new norms it decides whether or not to accept them (and modify its behaviour) [1], or reject them (and suffer potential sanctions, but this is a macro level issue). Although the issue of perceiving and deciding on accepting a norm has seldom been considered in the literature, it is of considerable importance in open dynamic systems regulated by norms. For example, in real human societies, norms are not *inherent* to the world, but rather created by some relevant authority and then made known to the parties to which a norm applies under the expectation that these norms will be accepted and complied with, which is not always the case. Furthermore, when an agent enters a new environment, it is possible that it will encounter a set of norms different from those for which it was designed. Here, an agent must be made aware of these norms and change its behaviour accordingly. Although the decision to accept or reject a norm is important, and must consider the future implications of ignoring a norm and later violating it, we do not detail it due to space limitations. Nevertheless, this may require subsequent obligations to remedy violations [10] or incur penalties, for example.

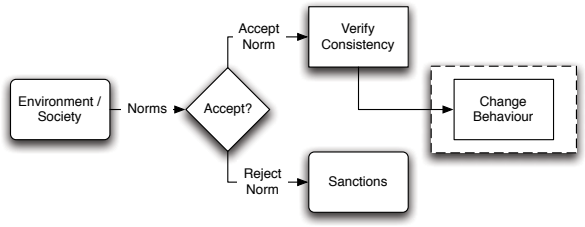

Figure 1: Overview of norms processing in our system.

We use the distinction of perception and belief to denote whether or not a norm has been accepted. If an agent has added a perceived norm to its belief-base, we consider the agent to have implicitly accepted the norm, and behaviour modification must ensue. In order to facilitate the integration of deontic states into an agent system, we adopt the view of Pörn [12] that deontic statements themselves can be seen as states of affairs. Thus, from an agent's perception point of view, the fact that a book is over a table (*e.g.* $over(book, table)$) is no different than a norm obliging an agent to drive on the right (*e.g.* $norm(X, Y, obligation(drive(right)))$). Given this representation, the difference between perception and belief indicates a norm's status in terms of acceptance by an agent.

After accepting a norm, an agent needs to make sure that the norm is not in conflict with the set of norms already in effect for that agent and then modify its behaviour. For example, if an agent accepts a norm n_1 that prohibits it from performing an action act, and later perceives a norm n_2 that obliges it to perform action act, then n_1 and n_2 are in conflict. Thus, it is necessary for an agent to make sure it does not accept conflicting norms which would result in inconsistent behaviours, such as creating and executing plans that carry out prohibited actions.

These processes are illustrated in Figure 1, in which rectangles represent agent reasoning processes, rounded rectangles represent environmental or societal processes, and diamonds represent decisions within an agent. The flow starts with the receipt of norms from the environment and the decision to accept or reject a norm. The processes that follow from this decision (including potential sanctions) consist of verifying consistency and subsequently changing behaviour. In this paper, we focus only on the behaviour modification part at the end of the flow in the figure. In particular, we avoid the issue of maintaining norm consistency, a very complex problem in itself, having been addressed by other efforts [14] that can be easily integrated into ours, as we discuss in our conclusions.

The point here is that while norms have been considered more generally, especially in the context of normative bodies within societies, the impact of norms on agent architectures, in particular BDI architectures, has received little attention. Specifically, the impact of norms at the level of an agent's control cycle has been largely overlooked. Therefore, building on previous efforts, in particular the planning capabilities described by Meneguzzi and Luck [9], and the communication framework defined by Vieira *et al.*[16], we address norms from the point at which an agent receives them.

In the following sections we proceed to detail the expected actions an agent must undertake in order to comply with norms under various combinations of activation and expiration conditions.

3. NORM PROCESSING

As we have seen, our norm representation includes conditions for norm activation and expiration, denoting when these norms are to affect an agent's behaviour. The main consequence of these conditions when modifying an agent's behaviour lies in what must be done in reaction to the agent accepting new norms. The two most evident situations regarding activation and expiration conditions are: when the activation condition is already true, which means that the norm must be enacted immediately and any contrary behaviours stopped; and when an expiration condition is already true, which means that the norm has expired and can be ignored. Clearly, there are a number of possible combinations of activation and expiration conditions, and each combination results in a different set of actions an agent must carry out in order to comply with the norm. We therefore analyse these combinations in Section 3.1 providing an overview of the outcome of each combination for an agent wanting to comply with the norm. Subsequently, we further detail algorithms that generate these outcomes, starting with algorithms to react to the activation of a norm in Section 3.2, followed by the results of norm expiration in Section 3.3.

3.1 Norm Outcomes

We summarise all norm condition combinations and their outcomes in Table 2, for each type of deontic modality considered in this paper (*i.e.* obligation and prohibition), and for each type of target for the modality (either an action or a world state). The activation and expiration condition columns give the truth-value of these conditions at the time the agent accepts the norm, so that (following our monotonic assumption) *True* means the expression is true and will remain so, while *False* means the expression is not yet true.

The first case we consider for all deontic modalities is when both the activation and the expiration condition are true, which results in the norm being ignored, as its expiration condition has already elapsed. Norms are also ignored when the activation condition is false but the expiration condition is true since, again, the norm has already expired. Now, if the activation condition is already true when an agent accepts an obligation, it must react to either achieve the world state specified in an $obligation(p)$, or execute the action specified in an $obligation(a)$.

If this same combination of conditions occurs for a prohibition, it means that an agent must refrain from achieving the offending world state or executing the offending action. Since an agent might have already adopted an offending plan, it must not only suppress plans that might violate the prohibition, but also drop any instances of these plans adopted as intentions.

Finally, when both the activation and expiration conditions are false, an agent must create new plans to enforce compliance as soon as the activation condition holds and add them to the plan library. This plan creation step is necessary because in this situation the norm has not yet been activated, but the agent must be prepared for its activation in the *future*. In particular, if the norm refers to an obligation to achieve a world state, the agent must create a plan that achieves the specified world state whenever the activation condition holds, whereas if it refers to an obligation to execute an action, the new plan must execute that action. (Note that plans to comply with obligations to achieve world states are created using some kind of planning capability, of which there are many, and we will not consider this further here.) Furthermore, if the norm refers to a prohibition to achieve a world state, the new plan must suppress all plans that achieve the prohibited world state whenever the activation condition holds, whereas if it refers to a prohibition to execute an action, the new plan must suppress all plans containing the prohibited action. In both these cases, not only must plans be suppressed, but violating intentions must also be dropped.

Now, we understand that creating new plans to comply with newly accepted norms, and later removing these plans after a norm has expired, from a pragmatic point of view, is not necessarily the

Deontic Modality	Activation Condition	Expiration Condition	Outcome
$obligation(O)$	True	True	Ignore norm
	True	False	Adopt plan to achieve O (if O is a world state) or adopt plan to execute O (if O is an action)
	False	False	Add plan to achieve O (if O is a world state) or add plan that includes O (if O is an action) to PL when activation holds
	False	True	Ignore norm
$prohibition(P)$	True	True	Ignore norm
	True	False	Drop intentions to achieve P and suppress plans that achieve P (if P is a world state) or drop intentions that include P and suppress plans that include P (if P is an action)
	False	False	Add plan to suppress plans that achieve P (if P is a world state) or plans that include P (if P is an action) when activation holds
	False	True	Ignore norm

Table 2: Combinations of activation and expiration conditions and their outcomes.

optimal solution. For example, it is certainly possible for an agent to use existing plans in its plan library to accomplish an obligation, avoiding the need to generate a new plan from scratch. However, an agent must still ensure that the plan's invocation condition is compatible with the activation condition of the obligation, as well as ensuring that it does not also bring about undesirable effects. This analysis process is complex in itself, and it is unclear how the cost of this process compares to planning from first principles. Furthermore, removing plans makes perfect sense under our monotonic assumption, since a norm will never occur in the exact same form again in the future. However, if we drop the monotonic assumption, plans should no longer be simply deleted after the norm has expired, but must instead be suppressed and stored for reuse for norms with multiply recurring activation and expiration conditions.

3.2 Norm Activation

Now that we have considered the possible high-level outcomes for an agent seeking to comply with norms, we consider how to deal with the receipt of obligations and prohibitions in more detail. In what follows, we assume only a BDI-type language with constructs for *goals to be* or *goals to do* or both; and a plan library or similar construct to store plans, which can be modified to reflect the set of possible plans an agent may adopt. This plan library is the target of our norm processing mechanism, which generates changes in the set of possible plans an agent may adopt either by adding new plans to comply with obligations or by preventing existing plans from being executed to comply with prohibitions. To accomplish these changes, we need to modify an agent's reasoning ability to be able not only to deal with the world through its actions, but also to deal with its own data structures, its available plans in particular. As a consequence, we need meta-level operators that can suppress plans from being selected from a plan library or from being generated by a planner, and can introduce new plans to the plan library.

In the following sections we refer to some algorithms as *templates*, or abstract *algorithms* that need to be further specified at runtime in order to be instantiated as concrete *algorithms*. This is because most agent languages define agent behaviours in terms of reactions to certain conditions in the environment, and the norm processing behaviours we specify are defined in terms of their activation and expiration conditions. More specifically, we consider the addition of plans to handle activation conditions for each type of deontic modality in the algorithms shown in Sections 3.2.1 and 3.2.2, and the plans to handle expiration conditions in Section 3.3.

Now, in order to illustrate the operation of these algorithms, we use an AgentSpeak(L) implementation as an example, instantiating the algorithms appropriately. For this to be meaningful, we must introduce the notation used for AgentSpeak(L) plans. An AgentSpeak(L) plan $e : b_1 \& \ldots \& b_m \leftarrow h_1; \ldots; h_n.$ is composed of a triggering event e, a context condition expressed by the $b_1, \ldots, b_m$ belief literals, and a list $h_1, \ldots, h_n$ of goals or actions [13]. Triggers may consist of the addition (denoted by a leading plus sign) or deletion (denoted by a leading minus sign) of beliefs, achievement goals (denoted by an exclamation mark), or queries (denoted by a question mark), so that the addition of an achievement goal `g` is denoted as `+!g`. In addition, we need to differentiate actions (which can also include *internal actions* representing some functionality internal to an agent) from beliefs, which we do with a preceding dot symbol; then, the action to vacuum a room is represented as $.vacuum(room)$ (and the action to remove a plan from the plan library is written $.remove\_plan(Plan)$), whereas the belief that a room is clean is represented as $clean(room)$. (Note that the specifics of how particular internal actions work are not important at this point, and we discuss them in detail later in Section 4 along with other implementation concerns.) Finally, we adopt the plan labelling convention used in the Jason interpreter [2], whereby plans are given unique labels in the form of predicates preceded by the *at* (@) sign. For example, a plan `@id e : c <- a.` is uniquely identified by the label `id`, which can be used later for plan manipulation operations referring to the plan `e : c <- a.`.

3.2.1 Obligations

When an agent chooses to comply with a newly perceived obligation, the steps it must take are as stated in more detail in Algorithm 1. While this is an algorithm, it is also a plan, since the algorithm must be encoded as a plan for an agent to be able to use it. The plan is very simple and follows the basic requirements of the outcomes of accepting an obligation as shown in Table 2. That is, if an obligation has already expired, it can be ignored; otherwise, new plans must be added to handle the activation and expiration conditions of the norm. If the obligation has been activated, it must be acted upon. Thus, Line 1 checks whether the agent believes the norm has expired, and if so, ignores it. Next in Line 4, the agent adds a new plan, from Algorithm 2, to deal with the activation condition. If the norm activation condition is true, the plan must be performed immediately, either achieving a specified world state or executing an action. Finally, the agent adds a new plan to handle the expiration of the obligation, which we explain in Section 3.3.

This newly added plan is detailed in Algorithm 2, and consists of either adding a new goal to achieve an obligatory world state, or executing an obligatory action upon the perception of the norm activation event. Note that Algorithm 2 is a plan *template*; that is, it shows the general form of plans that are instantiated with information about a specific norm and its associated activation condition. Since we are not concerned in this paper with the process by which an agent decides whether to comply with a norm, in both plans we assume that *acceptance* has been determined, indicated in the prerequisites of each algorithm. However, this decision to accept or reject a norm could be taken later in the reasoning process, depend-

Algorithm 1 Plan to comply with an obligation.

Require: Receipt of $norm(Activ, Exp, obligation(O))$
Require: Acceptance of $norm(Activ, Exp, obligation(O))$
Require: Belief Base BF; Plan Library PL
1: **if** $BF \models Exp$ **then**
2: **return**
3: **end if**
4: Create plan $L_{Activ,obligation(O)}$ using Algorithm 2 template
5: Add $L_{Activ,obligation(O)}$ to PL
6: **if** $BF \models Activ$ **then**
7: Execute plan from Algorithm 2
8: **end if**
9: Add plan from Algorithm 5 to deal with expiration

ing on the circumstances. In Algorithm 1, the requirements also refer to an agent's belief base, used to specify that the activation condition must hold before seeking to comply.

Algorithm 2 Plan template to react to activation of an obligation.

Require: Acceptance of $norm(Activ, Exp, obligation(O))$
Require: Receipt of $Activ$ event
Require: Plan is uniquely labelled with label $L_{Activ,obligation(O)}$
1: **if** O is a world state p **then**
2: Add goal to achieve p
3: **else if** O is an action a **then**
4: Add goal to execute a
5: **end if**

Example To illustrate these algorithms (plans), we use the example of a cleaner agent that is capable of using a vacuum cleaner to clean some room. Suppose that the cleaner agent accepts a norm to achieve a world state in which the floor is clean at *8:00* hours everyday, until Christmas Day, expressed as `+norm(time(800), day(xmas), obligation(clean(floor)))[source(env)]`. In our solution, this results in the generation of two plans, one associated with the activation condition, which leads to the adoption of a plan to achieve the obliged world state, and one associated with the expiration condition, which leads to the removal of all the plans associated with the specified norm. The two plans, using our notation introduced earlier, are illustrated in Listing 1. (While we give them here as illustration, we delay further explanation of the implementation of this example until Section 4.) The first plan, labelled `obligationStart(clean(floor))`, has the activation condition `time(800)` as its trigger, which leads to the adoption of an achievement goal `!clean(floor)`, (and in turn a plan) assumed to achieve a state in which `clean(floor)` holds. This goal addition, shown in Line 3 of Listing 1, corresponds to Line 2 of the generic Algorithm 2. The second plan, labelled `obligationEnd(clean(floor))` has the expiration condition `day(xmas)` as its trigger, and results in both plans being removed from an agent's plan library through two invocations of the `.remove_plan` internal action. Note that the steps to remove the plans associated with the obligation's expiration may seem self-referential but, due to the AgentSpeak reasoning cycle, on execution they are loaded into an instantiated intention that removes the original plans from the plan library.

Conversely, when an agent is obliged to execute an action, a similar plan schema is generated, but instead of adopting a goal to achieve a specified state, the new plan needs simply to execute the specified action. Thus, if a cleaner accepts a norm to vacuum the floor at `8:00` hours every day, until Christmas day, expressed as `+norm(time(800), day(xmas), obligation(.vacuum(floor)))[source(env)]`, very similar plans are adopted, but instead of adding an achievement goal to its intention structure, the specified action is executed.

```
1 @obligationStart(clean(floor))
2 +time(800) : true
3  <- !clean(floor).
4
5 @obligationEnd(clean(floor))
6 +day(xmas) : true
7  <- .remove_plan(obligationStart(clean(floor)));
8    .remove_plan(obligationEnd(clean(floor))).
```

Listing 1: Plans for the clean state norm.

3.2.2 Prohibitions

Now, when the accepted norm refers to a prohibition, we need to use Algorithm 3. This consists of first checking for norm expiration, then adding a plan to handle norm activation, and checking whether or not the norm has already been activated, in which case the added plan must be executed.

Algorithm 3 Plan to comply with a prohibition.

Require: Receipt of $norm(Activ, Exp, prohibition(P))$
Require: Acceptance of $norm(Activ, Exp, prohibition(P))$
Require: Belief base BF
1: **if** $BF \models Exp$ **then**
2: **return**
3: **end if**
4: Create plan $L_{Activ,prohibition(P)}$ using Algorithm 4 template
5: Add $L_{Activ,prohibition(P)}$ to PL
6: **if** $BF \models Activ$ **then**
7: Execute plan from Algorithm 4
8: **end if**
9: Add plan from Algorithm 6 to deal with expiration

The plan to deal with prohibition activation, detailed in Algorithm 4, consists of first scanning an agent's intentions (*i.e.* the plans already adopted) for instances of the prohibited action, or for plans having a prohibited world state as a consequence. If plans violating the prohibition are found as intentions, they must be dropped immediately. Afterwards, the plan library is similarly scanned for plans that may violate the prohibition if executed. If the prohibition refers to a state, all plans with this state as a consequence must be suppressed, while if the prohibition refers to an action, all plans with this action must be suppressed. In addition, suppressing the plans, suppressed plans are stored in the set $S_{Plans,prohibition(P)}$, so that when the prohibition expires later, these plans can be restored. Like Algorithm 2, the plan of Algorithm 4 is a plan *template* (that is, a general form of plan that becomes concrete after the information about a specific norm and its activation and expiration conditions is known).

It is important to note here that we leave the meaning of plan suppression relatively ambiguous, since the details of how this is done depend on the particular way in which an agent architecture operates. Later in this paper we show how this can be achieved in an AgentSpeak(L)-type agent.

Example If our cleaning agent was prohibited from entering a room with classified documentation, expressed as `+norm(time(800), day(xmas), prohibition (in(classifRoom)))[source(env)]`, two new plans need to be generated. First, for activation, we need to make sure that the plans that result in the cleaner being in the classified room are *suppressed* from execution, as shown in Listing 2. In this plan, Lines 3 to 5 of Listing 2 correspond to Lines 13 to 15 of the generic Algorithm 4. Moreover, when the prohibition expiration condition becomes true, not only do the plans to handle the activation and expiration conditions need to be removed, but also the plans that were suppressed by the activation condition need to

Algorithm 4 Plan template to react to state prohibition activation.

Require: Acceptance of $norm(Activ, Exp, prohibition(P))$
Require: Receipt of $Activ$ event
Require: Intention structure I; Plan library PL
Require: Plan uniquely labelled with label $L_{Activ,prohibition(P)}$
Ensure: Suppressed plans are stored in set $S_{Plans,prohibition(P)}$

for all Intention $i \in I$ **do**
  if (P is a world state p) and (p is a consequence of i) **then**
    Drop intention i
  else if P is an action a **then**
    for all Steps s in remaining steps of i **do**
      if $s = a$ **then**
        Drop intention i
      end if
    end for
  end if
end for
for all Plans $pl \in PL$ **do**
  if (P is a world state p) and (p is a consequence of i) **then**
    Suppress pl
    $S_{Plans,prohibition(P)} = S_{Plans} \cup pl$
  else if P is an action a **then**
    for all Steps s in pl **do**
      if $s = a$ **then**
        Suppress pl
        $S_{Plans,prohibition(P)} = S_{Plans} \cup pl$
      end if
    end for
  end if
end for

be *unsuppressed*.

```
1  @prohibitionStart(in(classifRoom))
2  +!Start : true
3   <- !findPlansWithEffect(in(classifRoom), SPlans);
4      !suppressPlans(SPlans);
5      +suppressedPlans(in(classifRoom),SPlans).
6
7  @prohibitionEnd(in(classifRoom))
8  +!End : suppressedPlans(in(classifRoom),SPlans)
9   <- !unsuppressPlans(SPlans);
10     .remove_plan(prohibitionStart(in(classifRoom)));
11     .remove_plan(prohibitionEnd(in(classifRoom))).
```

Listing 2: Plans generated from a state prohibition.

Plans to effect restrictions on executing actions are very similar to those relating to achieving world states, the only difference being in the process for selecting the plans that need to be suppressed. In this case, the plans searched for are those that contain a particular action. For example, if the cleaning agent might be obliged not to vacuum a table during its rounds of cleaning through the norm `+norm(time(800), day(xmas), prohibition(vacuum(table)))[source(env)]`. We do not include the example plans due to space constraints, but they should be obvious.

3.3 Norm expiration

Now that we have seen the plans needed to start complying with norms under several circumstances, we need to examine how an agent behaviour is modified as a result of a norm expiring. When an agent accepts a norm and changes its behaviour as a result of the norm becoming active, it either includes extra plans to comply with obligations or suppresses some of its plans in order to violate a prohibition. However, these behaviour modifications should not become permanent within an agent if the norms that caused them cease to be active. Moreover, our monotonicity assumption entails that once a norm has been activated and then expired, it will never become active again. Thus, Algorithms 1 and 3, containing plans for reacting to norms, also include a final step to add a plan dealing with norm expiration to the plan library. Such norm expiration plans aim to *undo* the behavioural changes effected when the norms were activated, thus restoring the plan library to a state in which an agent's behaviour is not affected by them. Thus, the plan in Algorithm 5 consists both of removing the plan responsible for dealing with obligation activation and afterwards of removing itself from an agent's plan library. Both these plans must be individually identifiable within an agent's plan library, so we label them respectively $L_{Activ,obligation(O)}$ and $L_{Exp,obligation(O)}$ in order to remove them when they are no longer needed.

Algorithm 5 Plan to react to the expiration of an obligation.

Require: Acceptance of $norm(Activ, Exp, obligation(O))$
Require: Receipt of Exp event
Require: Label $L_{Activ,obligation(O)}$ for a norm activation plan
Require: Plan library PL
Ensure: Plan is uniquely labelled with label $L_{Exp,obligation(O)}$

1: Remove plan $L_{Activ,obligation(O)}$ from PL
2: Remove plan $L_{Exp,obligation(O)}$ from PL

The acceptance of prohibitions, on the other hand, not only adds new plans to react to norm activation and expiration, it also affects which plans are available to an agent after a prohibition has been activated. Thus, the plan to react to the expiration of a prohibition must not only remove the new plans added to comply with the norm, it must also restore the plans previously suppressed to their initial state of availability. The plan of Algorithm 6 accomplishes this by unsuppressing the initially suppressed plans which, in the plan of Algorithm 4, were stored in the set $S_{Plans,prohibition(P)}$, and then removing plans $L_{Activ,prohibition(P)}$ and $L_{Exp,prohibition(P)}$ from the plan library.

Algorithm 6 Plan to react to the expiration of a prohibition

Require: Acceptance of $norm(Activ, Exp, prohibition(P))$
Require: Receipt of Exp event
Require: Label $L_{Activ,prohibition(P)}$ for a norm activation plan
Require: Plan library PL
Require: $S_{Plans,prohibition(P)}$ of suppressed plans
Ensure: Plan is uniquely labelled with label $L_{Exp,prohibition(P)}$

1: Unsuppress all plans from $S_{Plans,prohibition(P)}$
2: Remove plan $L_{Activ,prohibition(P)}$ from PL
3: Remove plan $L_{Exp,prohibition(P)}$ from PL

4. NORMATIVE AGENTSPEAK(L)

In order to test the viability of our solution in a practical agent language, we have developed an implementation of the strategies outlined in Section 3 using an AgentSpeak(L) interpreter. An important part of this involves the manipulation of an agent's own plan library, necessitating a means to perform meta-reasoning, allowing AgentSpeak(L) plans to manipulate other plans. With such a meta-reasoning facility in place, we can create AgentSpeak(L) plans that accomplish the norm-induced behaviour modification described above. We also point out that, while the plans shown in Section 3 use constructs that were not described in detail, this section clarifies all the plan constructs used throughout the paper.

4.1 Meta-reasoning for AgentSpeak(L)

The AgentSpeak(L) language does not have explicit constructs for the analysis of a plan library, yet this is required in the strategies described in Section 3 and implemented in Section 4.2. In particular, for an agent to evaluate its existing behaviours, encoded in

Action	Effect
`.plan_steps(P,S)`	takes a plan P and unifies its plan steps as a list of literals with S
`.plan_conseq(P,C)`	takes a plan P and unifies its declarative consequences with C
`.action(A)`	succeeds if A refers to an action
`.literal(L)`	succeeds if L to a literal
`.remove_plan(P)`	removes P from the plan library
`.suppress_plan(P)`	suppresses the specified plan from being executed
`.unsuppress_plan(P)`	allows a previously suppressed plan to be executed

Table 3: Summary of the meta-reasoning actions

its plans, we require the introduction of meta-reasoning operators that allow regular AgentSpeak(L) plans themselves to explore and process other plans in the plan library. In our system, we construct such operators, as summarised in Table 3, through the use of *internal agent actions*. The common understanding of agent actions is that they are environment transformation operators, so that when an agent invokes an action, some consequence in the environment is expected. However, when some custom computation needs to take place within a single reasoning cycle, Bordini *et al.*use the concept of an *internal action* [2]. This allows an agent to access extensible libraries of custom procedures that can be executed instantaneously by an agent. Unlike traditional actions, internal actions do not cause changes in the environment, and since they are executed instantaneously, they can be included in either the body or the context of a plan, to refine the process of selecting applicable plans. Internal actions are denoted by a preceding dot, so the internal action to suppress a plan is represented as `.suppress_plan(Plan)`.

Most of the meta-level actions of Table 3 either have simple outcomes or implement parts of the algorithms described in Section 3. However, the first two actions in the table are needed specifically to deal with the way in which AgentSpeak(L) operates, and we need to clarify them further. Plans to ensure compliance with prohibitions are more complex in that they require an agent to scan its entire plan library looking for violating plans. For prohibitions relating to executing an action, this requires finding all plans in the plan library that contain the prohibited action and suppressing their execution. This is shown in Algorithm 7.

Algorithm 7 Find plans with action.

Require: AgentSpeak plan library PL
Require: Action act
Ensure: A list PL_A of plans containing act
1: **for all** Plans $\{t : c \leftarrow b.\} \in PL$ **do**
2: **for all** Steps $s_i \in b$ **do**
3: **if** s_i unifies with act **then**
4: Add $\{t : c \leftarrow b.\}$ to PL_A
5: **end if**
6: **end for**
7: **end for**
8: **return** PL_A

Prohibitions relating to achieving certain world states require an agent to analyse the effects of each of the plans in its plan library, and suppress the execution of those that have the prohibited state as an effect. An algorithm to accomplish this is shown in Algorithm 8.

4.2 AgentSpeak plan modification mechanisms

Using these internal actions, we can create AgentSpeak(L) plans that add newly accepted norms, as described in Section 2.3, to the set of active norms. As we have seen, adding obligations is relatively straightforward, and we omit the meta-level plans for their

Algorithm 8 Find plans with effect.

Require: AgentSpeak plan library PL
Require: Proposition p
Ensure: A list PL_P of plans containing p
1: **for all** Plans $\{t : c \leftarrow b.\} \in PL$ **do**
2: Get the effects E of $\{t : c \leftarrow b.\}$
3: **for all** Effects $e \in E$ **do**
4: **if** e unifies with p **then**
5: Add $\{t : c \leftarrow b.\}$ to PL_P
6: **end if**
7: **end for**
8: **end for**
9: **return** PL_P

```
1  @prohibitionStart(Prohibition)
2  +!Start : true
3   <- !findPlansWithAction(Prohibition, SPlans);
4     !suppressPlans(SPlans);
5     +suppressedPlans(Prohibition,SPlans).
6
7  @prohibitionEnd(Prohibition)
8  +!End : suppressedPlans(Prohibition,SPlans)
9   <- !unsuppressPlans(SPlans);
10    .remove_plan(prohibitionStart(Prohibition));
11    .remove_plan(prohibitionEnd(Prohibition)).
```

Listing 3: Template plans generated from an action prohibition.

addition, focussing instead on those responsible for handling the addition of prohibitions. When a prohibition referring to an action is added, we need to create two plans, one to handle the start of the prohibition and another to handle the end of the prohibition.

Thus, when a prohibition on an action becomes effective, an agent needs first to find all plans with the prohibited action and then suppress each of the plans containing the offending action. Finding these plans involves a `!findPlansWithAction( Prohibition, SelPlans)` plan, which uses the `.plan_steps(Plan,Steps)` action to explore each plan step, corresponding to step 2 of Algorithm 7. Once the plans are identified, the plan needs to suppress these plans with the `!suppressPlans(SelPlans)` plan, which uses the `.suppress_plan(Plan)` action. In our implementation, plans that are suppressed are removed from the plan library, and thus not considered as options to achieve a certain goal. It is also important to keep track of the suppressed plans so that they can be unsuppressed later. When the prohibition ceases to be effective, we need to unsuppress the plans previously suppressed, as well as removing the plans related to this particular norm, since they will no longer be necessary. A meta-plan responsible for creating these norm-related plans uses the start and end conditions to create triggers for two plans that accomplish the suppression and unsuppression of the necessary plans, generating plans using the template in Listing 3.

To add the plans that handle prohibitions on world states, the necessary steps are similar to those for prohibitions on actions, the only difference being that the search criterion for offending plans involves the effects of these plans. We extract the effects of plans using the `.plan_conseq(Plan, Consequences)` action, used in the `!findPlansWithEffect( Prohibition, SelPlans)` plan. Like in the prohibition for actions, a meta-plan responsible for creating these norm-related plans creates two plans following the template shown in Listing 4.

5. CONCLUSIONS

In this paper, we have described a framework of concrete behaviours for classical agent languages that enable them to effect changes in their own plan libraries to conform to new norms ac-

```
1  @prohibitionStart(Prohibition)
2  +!Start : true
3   <- !findPlansWithEffect(Prohibition, SPlans);
4      !suppressPlans(SPlans);
5      +suppressedPlans(Prohibition,SPlans).
6
7  @prohibitionEnd(Obligation)
8  +!End : suppressedPlans(Prohibition,SPlans)
9   <- !unsuppressPlans(SPlans);
10     .remove_plan(prohibitionStart(Prohibition));
11     .remove_plan(prohibitionEnd(Prohibition)).
```

Listing 4: Template plans generated from a state prohibition.

cepted from the environment. Our framework is sufficiently generic that it can be extended into any traditional BDI style agent language. Importantly, we have also developed these general algorithms further into a concrete instantiation in AgentSpeak(L) (using a new toolkit of meta-reasoning operators, that has not been considered previously), providing a novel contribution in itself, as well as an illustration and realisation of the algorithms. We show how our framework can generate new plans to enable agents to comply with norms, and remove the plans when the norms are no longer relevant, through a series of examples throughout the paper, demonstrating the practicality of our approach.

In this work, we have adopted a stance that takes theoretical notions and integrates them into practical agent languages. Previous work has also addressed similar concerns, in moving from largely intractable deontic modalities into simpler, yet useful representations of norms to be used in a concrete system. For example, one of the first practical architectures for a norm-driven agent was Kollingbaum and Norman's NoA [6], which takes a BDI-like agent architecture and changes the focus of agent behaviour from achieving desires to fulfilling norms. As in NoA, we use an explicit representation of the effects of an agent's plans to detect potential norm violations, as well as deciding which plans are more suitable for achieving an obligation, but our agents are still driven by their desires like traditional BDI agents. In contrast, Vazquez-Salceda *et al.*[15] take the ISLANDER [3] formalism and use it to establish guidelines for the implementation of a normative system, its monitoring and the enforcement of its norms. Our work can be seen as complementary, since we provide the machinery to modify the behaviour of an agent willing to accept norms, as opposed to being concerned with the rest of the system. Unlike other approaches, such as electronic institutions [1], which typically *require* compliance, our agents may choose to ignore a norm, even if it may lead to potential penalties.

Due to the rather large scope of normative agents, we have not provided a detailed account of the important issue of maintaining the consistency of a set of norms. Other types of potential clashes involve overlapping of norm conditions, including their activation and expiration; for example, if an agent accepts a norm prohibiting work from time 12 to time 14, and another prohibiting work from time 11 to time 15, plans may be modified due to the activation of the second prohibition at time 11, and then modified again due to the expiration of the first prohibition at time 14, jeopardising the second prohibition in the process. However, it is important to point out that a solution for addressing this could be easily adapted from the work of Vasconcelos *et al.*[14], which provides an algorithm for resolving conflict and inconsistency in sets of norms using a unification-based technique. Similarly, our agents could resolve norm conflicts by adapting the mechanism used by Kollingbaum's [6] NoA architecture for the same purpose. Thus, by assuming an existing norm consistency maintenance process, we have reduced the problem we address to focus on individual norm additions and deletions, avoiding dilution of our efforts, and facilitating a more specific consideration of the relevant issues.

Acknowledgments: The first author is supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) of the Brazilian Ministry of Education.

6. REFERENCES

[1] H. Aldewereld, F. Dignum, A. García-Camino, P. Noriega, J. A. Rodríguez-Aguilar, and C. Sierra. Operationalisation of norms for usage in electronic institutions. In *Proc. 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 223–225, 2006.

[2] R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, 2005.

[3] M. Esteva, J. A. Padget, and C. Sierra. Formalizing a language for institutions and norms. In J.-J. C. Meyer and M. Tambe, editors, *Intelligent Agents VIII*, volume 2333 of *LNCS*, pages 348–366. Springer, 2001.

[4] N. Faci, S. Modgil, N. Oren, F. Meneguzzi, S. Miles, and M. Luck. Towards a monitoring framework for agent-based contract systems. In M. Klusch, M. Pechoucek, and A. Polleres, editors, *Cooperative Information Agents XII*, volume 5180 of *LNCS*, pages 292–305, 2008.

[5] A. J. I. Jones and I. Pörn. 'ought' and 'must'. *Synthese*, 66(1):89–93, 1986.

[6] M. J. Kollingbaum and T. J. Norman. Norm adoption and consistency in the NoA agent architecture. In *PROMAS 2003*, volume 3067 of *LNCS*, pages 169–186. Springer, 2003.

[7] A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.

[8] F. Lopez y Lopez, M. Luck, and M. d'Inverno. A normative framework for agent-based systems. In *Proc. 1st Int. Symp. on Normative Multi-Agent Systems*, 2005.

[9] F. Meneguzzi and M. Luck. Leveraging new plans in AgentSpeak(PL). In M. Baldoni, T. C. Son, M. B. van Riemsdijk, and M. Winikoff, editors, *Declarative Agent Languages and Technologies VI*, volume 5397 of *LNCS*, pages 63–78. Springer, 2008.

[10] N. Oren, M. Luck, and T. J. Norman. Argumentation for normative reasoning. In *Proc. Symp. Behaviour Regulation in Multi-Agent Systems*, pages 55–60, 2008.

[11] N. Oren, S. Panagiotidi, J. Vazquez-Salceda, S. Modgil, M. Luck, and S. Miles. Towards a formalisation of electronic contracting environments. In *Proc. 12th COIN Workshop*, pages 61–68, 2008.

[12] I. Pörn. *Logic of Power*. Blackwell Publishers, 1970.

[13] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. V. de Velde and J. W. Perram, editors, *Agents Breaking Away*, volume 1038 of *LNCS*, pages 42–55. Springer, 1996.

[14] W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman. Resolving conflict and inconsistency in norm-regulated virtual organizations. In *Proc. 6th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 1–8, 2007.

[15] J. Vázquez-Salceda, H. Aldewereld, and F. Dignum. Norms in multiagent systems: from theory to practice. *Computer Systems: Science & Engineering*, 20(4), 2005.

[16] R. Vieira, Á. F. Moreira, M. Wooldridge, and R. H. Bordini. On the formal semantics of speech-act based communication in an agent-oriented programming language. *Journal of Artificial Intelligence Research*, 29:221–267, 2007.