Nu-BDI: Norm-aware BDI Agents

Felipe Meneguzzi[1], Wamberto Vasconcelos[2], Nir Oren[2], and Michael Luck[3]

[1] Faculdade de Informática, PUCRS, Porto Alegre, Brazil
[2] Department of Computing Science, University of Aberdeen, Aberdeen, UK
[3] Department of Informatics, King's College London, London, UK
felipe.meneguzzi@pucrs.br, wvasconcelos@acm.org, nir.oren@abdn.ac.uk, michael.luck@kcl.ac.uk

Abstract Systems of autonomous and self-interested agents interacting to achieve individual and collective goals may exhibit undesirable or unexpected properties if left unconstrained. Using deontic concepts of obligations, permissions and prohibitions to describe, what must, may and should not be done, norms have been widely proposed as a means of defining and enforcing societal constraints. Recent efforts to provide norm-enabled agent architectures that limit plan choices suffer from interfering with an agent's reasoning process, and thus limit autonomy more than is required by the norms alone. In response, in this paper we describe an extension of the BDI architecture which enables normative reasoning, providing agents with a means to choose and customise plans (and their constituent actions), so as to ensure compliance with norms.

1 Introduction

Systems of autonomous and self-interested agents interacting to achieve individual and collective goals may exhibit undesirable or unexpected properties if left unconstrained. Norms have been proposed as a means of defining and enforcing constraints aimed at ensuring that such undesired behaviour is avoided [12,18]. Norms are generally specified using deontic concepts of obligations, permissions and prohibitions to identify, respectively, what must, may and should not be done so as to ensure certain system properties. Early work on normative systems focused on model-theoretic or philosophical aspects of deontic logics [26], but more recent work has addressed how norms may be more suitably represented in computational systems [20,21], their enforcement [13] and their impact on the society as a whole, abstracting away the details of mechanisms through which individual agents reason with and about norms and how individual behaviours are affected by norms [1,12,18].

However, practical normative systems require analysis and specification of the processes through which norms are recognised, decisions about whether to comply with them are taken, and behaviour is adjusted appropriately. Some recent efforts have sought to provide norm-enabled architectures [19,22] to constrain an agent's behaviour to comply with norms in terms of permitted or forbidden mental states. For example, compliance with an obligation to move to a certain

location limits an agent's choice of plans containing moving actions to only those in which the target of the actions is the obliged location. While such architectures capture this notion at a basic level, for example in preventing parts of a plan library from being adopted [22], or replacing the goals of an agent with the fulfilment of specific norms [19], they suffer from interfering with an agent's reasoning process, and thus limit autonomy more than required by norms alone.

In response, in this paper we introduce ν-BDI,[4] an extension of the BDI architecture [23] that enables normative reasoning, and provides a means for agents to choose and customise plans (and their constituent actions), so as to ensure compliance with norms. The paper makes three significant contributions in providing: fine-grained tailoring of plan restrictions; a plan annotation mechanism to identify violating plan instances, and limit their adoption; and a technique permitting the selective and incremental violation of norms in cases where goal achievement would not otherwise be possible.

We start by reviewing the BDI agent model and introducing a basic interpreter in Section 2. In Section 3 we introduce the notation used for precisely specifying normative restrictions, including restrictions over acceptable domains. Using this, we develop in Section 4 an agent architecture capable of reasoning with these norms, thus affecting specific plan instances that are adopted, deciding on norm compliance as plan instances are selected. In doing so, we fulfil the need for pragmatic normative agent architectures capable of filtering norm compliant plans and deciding upon them. We review related work in Section 5, finally drawing conclusions in Section 6.

2 Preliminaries

In this section we review the well-known BDI architecture, which is the foundation of our norm-aware architecture.

In order to explain the operation of our agent interpreter, we need to introduce some notation and definitions. We use first-order constructs for various elements of the agents and norms.

Definition 1 (Term). *A term, denoted generically as τ, is a variable w, x, y, z (with or without subscripts), a constant a, b, c (with or without subscripts) or $f^n(\tau_0, \ldots, \tau_n)$, that is, an n-ary function f^n applied to (possibly nested) terms $\tau_0, \ldots, \tau_n$.* □

Definition 2 (Predicate). *A predicate (or a first-order atomic formula), denoted as φ, is any construct of the form $p^n(\tau_0, \ldots, \tau_n)$, where p^n is an n-ary predicate symbol applied to terms $\tau_0, \ldots, \tau_n$. A first-order atomic formula, denoted as Φ, is defined as $\Phi ::= \Phi \wedge \Phi | \neg\Phi | \forall x.\Phi | \varphi$.* □

We assume the usual abbreviations: $\Phi \vee \Phi'$ stands for $\neg(\neg\Phi \wedge \neg\Phi')$, $\exists x.\Phi$ stands for $\neg\forall x.\neg\Phi$, $\Phi \rightarrow \Phi'$ stands for $\neg\Phi \vee \Phi'$ and $\Phi \leftrightarrow \Phi'$ stands for $(\Phi \rightarrow \Phi') \wedge (\Phi' \rightarrow$

[4] ν-BDI, is a pun involving the Greek letter ν we use to refer to norms and the English word "'new", the approximate pronunciation of ν in English.

Φ). Additionally, we also adopt the equivalence $\{\Phi_1, \ldots, \Phi_n\} \equiv (\Phi_1 \wedge \cdots \wedge \Phi_n)$ and use these interchangeably. In our mechanisms we use first-order unification [11] which is based on the concept of substitutions.

Definition 3 (Substitution). *A substitution σ is a finite and possibly empty set of pairs x/τ, where x is a variable and τ is a term.* □

The application of a substitution is defined as:

1. $c \cdot \sigma = c$ for a constant c.
2. $x \cdot \sigma = \tau \cdot \sigma$ if $x/\tau \in \sigma$; otherwise $x \cdot \sigma = x$.
3. $p^n(\tau_0, \ldots, \tau_n) \cdot \sigma = p^n(\tau_0 \cdot \sigma, \ldots, \tau_n \cdot \sigma)$.

Unifications can be *composed*; that is, for any $\sigma_1 = \{x_1/\tau_1, \ldots, x_n/\tau_n\}$ and $\sigma_2 = \{y_1/\tau'_1, \ldots, y_k/\tau'_k\}$, their composition, denoted as $\sigma_1\sigma_2$, is defined as $\{x_1/(\tau_1 \cdot \sigma_2), \ldots, x_n/(\tau_n \cdot \sigma_2), z_1/(z_1 \cdot \sigma_2), \ldots, z_m/(z_m \cdot \sigma_2)\}$, where $\{z_1, \ldots, z_m\}$ are those variables in $\{y_1, \ldots, y_k\}$ that are not in $\{x_1, \ldots, x_n\}$. A substitution σ is a *unifier* of two terms τ_1, τ_2, if $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$.

Definition 4 (Unify Relation). *unify(τ_1, τ_2, σ) holds iff $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$, for some σ. unify$(p^n(\tau_0, \ldots, \tau_n), p^n(\tau'_0, \ldots, \tau'_n), \sigma)$ holds iff unify$(\tau_i, \tau'_i, \sigma), 0 \leq i \leq n$.* □

Two terms τ_1, τ_2 are related through the *unify* relation if there exists a substitution σ that makes the terms logically equivalent. We denote as $\bar{\varphi}$ a first-order predicate whose terms are either constants or variables associated (via a substitution) with constants. Here, we adopt Prolog's convention [3] and use strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants. We assume the availability of a sound and complete first-order inference mechanism which decides if Φ' can be inferred from Φ, denoted as $\Phi \vdash \Phi'$. In this paper we use a mechanism to determine if a formula Φ can be inferred from a set of ground predicates, and, if so, under which substitution; that is, $\{\bar{\varphi}_0, \ldots, \bar{\varphi}_n\} \vdash \Phi \cdot \sigma$.

In this work we consider an abstract BDI interpreter, inspired by the dMARS architecture [10]. We define an agent in terms of its information model as follows.[5]

Definition 5 (Agent). *An agent is a tuple $\langle Ag, Ev, Bel, Plib, Int \rangle$, where Ag is the agent identifier, Ev is a queue of events, Bel is a belief base, $Plib$ is a plan library, and Int is an intention structure.* □

Recently perceived events are stored in a queue and ordered by arrival time. An event may be a belief addition or deletion, or a goal addition or deletion. Belief additions are *positive ground predicates* perceived as true, and belief deletions are *negative ground predicates* perceived as false. Goal additions indicate new goals posted, and goal deletions represent goals dropped for some reason.

[5] We use Greek letters to denote elements of the underlying logic system as well as norms, and use Latin letters to denote elements of the agent interpreter.

Definition 6 (Events). *An event queue Ev is composed of ground first-order predicates representing events* $[e_1, \ldots, e_n]$ *ordered by occurrence time. Events* e_i *can be one of four possible cases:* i) *a belief addition* $+\bar{\varphi}$*, whereby belief* $\bar{\varphi}$ *is added to the belief base;* ii) *a belief deletion* $-\bar{\varphi}$*, whereby belief* $\bar{\varphi}$ *is removed from the belief base;* iii) *a goal addition* $+!\bar{\varphi}$*, whereby the goal to achieve* $\bar{\varphi}$ *is posted to the agent; or* iv) *a goal deletion* $-!\bar{\varphi}$*, whereby the goal to achieve* $\bar{\varphi}$ *has been dropped by the agent.* □

The belief base comprises a set of logic predicates, which can be queried through an entailment relation, as follows.

Definition 7 (Beliefs). *A belief base Bel is a finite and possibly empty set of ground first-order predicates* $\{\bar{\varphi}_1, \ldots, \bar{\varphi}_n\}$*, with an associated logical entailment relation* $\vdash$ *for first-order formulae.* □

The plan library, defined below, stores the plans of action available. Each step in a plan body may be either an action (causing effects in the environment) or a subgoal (causing the addition of a new plan from the plan library to the intention structure).

Definition 8 (Plans). *A plan library Plib is a finite and possibly empty set of uninstantiated plans* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$*. Each plan* $\mathcal{P}_i$ *is a tuple* $\langle t, c, bd \rangle$ *where t is an invocation condition, namely, an event of the form* $+\bar{\varphi}, -\bar{\varphi}, +!\bar{\varphi}$*, or* $-!\bar{\varphi}$ *(cf. Definition 6), indicating the event that causes the plan is to be adopted, c is a context condition in the form of a first-order formula over the agent's belief base, and bd is a body consisting of a finite and possibly empty sequence of steps* $[s_0, \ldots, s_n]$*, with each* s_i *representing either the invocation condition of a plan, or an action (cf. Definition 9).* □

Actions are identified by first-order atomic formulae, and in order to deal with declarative world-states we must define what an action entails, so that we are able to use action execution as the target of normative stipulations in Section 3. Thus, we define an action's consequences in Definition 9, as follows:

Definition 9 (Action). *An action is a tuple* $\langle \varphi, \varpi, \varepsilon \rangle$ *where*

- φ *is an action identifier, represented as a first order predicate* $p^n(\tau_0, \ldots, \tau_k)$ *with variables* $\tau_0, \ldots, \tau_l$*.*
- ϖ *is the action's precondition, represented as a formula* Φ *containing first order predicates* $\varphi_0^p, \ldots, \varphi_m^p$*, which in turn contain variables* $\tau_{l+1}, \ldots, \tau_n$*.*
- ε *is a set of first order predicates representing the effects of the action.* ε *is composed of two sets,* ε^+ *and* ε^-*. These sets represent new beliefs to be added to the belief base (if they are members of* ε^+*), or beliefs to be removed from the belief base (if they are members of* ε^-*).*

All variables in the predicates of ε *must be contained in the set of variables* $\tau_0, \ldots, \tau_n$ *used in the action identifier and preconditions. We refer to an action by its identifier,* φ*. We refer to the preconditions of an action* φ *as* $\varpi(\varphi)$*, and its effects as* $\varepsilon(\varphi)$*. We refer to the set of all possible actions as Actions.* □

Within an agent's plans action invocations and belief modifications are both represented as first order predicates. However, within the body of a plan, belief predicates only appear associated with the symbols for addition and deletion (cf. Definition 8), denoting updates to the belief base. Conversely, predicates referring to actions (*i.e.* their identifier, cf. Definition 9) appear on their own within the body a plan denoting that an action is to be executed, as shown in Example 1.

Example 1. The plan of our scenario is represented as follows:

$$\left\langle +level(X, medium), (high\_risk(X)), \begin{bmatrix} isolate(X), \\ evacuate(X, Y), \\ reroute(X, Z) \end{bmatrix} \right\rangle$$

That is, if a belief $level((X), medium)$ is added to the belief base, stating that the level of emergency of area X is *medium*, and X is a *high_risk* area, then the plan is to:

i) *isolate* area X (thus preventing anyone from entering the area);
ii) *evacuate* the area, moving everyone from X to Y, and
iii) *reroute* traffic through X to go through Z. ■

Finally, the intention structure comprises the agent's intentions, each of which contains partially instantiated plans to be executed by the agent.

Definition 10 (Intentions). *An intention structure Int is a finite and possibly empty set of intentions* $\{int_1, \ldots, int_n\}$. *Each* int_i *is a tuple* $\langle \sigma, \bar{st} \rangle$, *where* σ *is a substitution and* $\bar{st}$ *is an intention stack (containing the steps remaining to be executed to achieve the intention).* □

The specification above provides a minimal information model required for BDI agent execution. This model is used with little or no modifications in various agent interpreters such as PRS [16], Jason [4] and others. However, these interpreters have no mechanism for normative processing, relying on a designer to hard-code any norm-influenced behaviour. As discussed in the introduction, an agent's ability to act while considering its norms provides an increase in the agent's capabilities and our goal is to enable BDI agents to become norm aware. An overview of the basic BDI interpreter is illustrated as the white boxes in Figure 1, together with our proposed additions represented as grey boxes. While we do not provide further details on the operation of the basic BDI interpreter, we refer the reader to the original description of this type of interpreter at [16]. Thus, in the following sections we specify the framework within which ν-BDI operates and the algorithms necessary for generalised normative behaviour.

3 Norm Representation

Using deontic concepts of obligations, permissions and prohibitions to describe, what must, may and should not be done, norms have been widely proposed as a means of defining and enforcing societal constraints. In this paper, since

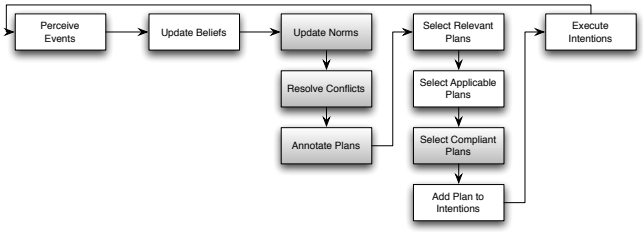

Figure 1: Control flow for the reasoning process

we are concerned with the impact of norms on reasoning and behaviour, we pay particular attention to the scope of influence of norms. In this respect, we consider two distinct means of addressing this: first, we draw on aspects similar to those presented in [12] and [19] in that our norms are *conditional*, both for activation (when they come into force) and expiration (when they cease effect), limiting application to periods of time; and second, we add *constraints* [17], limiting application to particular plans and actions, and ensuring that norms are not over-restrictive. In this section, we adapt and extend the notation for specifying norms of [25].

Definition 11 (Constraint). *Constraints, represented as γ, are constructs of the form $\tau \lhd \tau'$, where τ, τ' are first-order terms (that is, a variable, a constant or a function applied to terms) and $\lhd$ is one of the infix binary operators $=, \neq$, $>, \geq, <$, or $\leq$. A conjunction of constraints is denoted as $\Gamma = (\gamma_1 \wedge \cdots \wedge \gamma_n)$.* □

We use arithmetic functions to build terms τ. For example, $10 > \mathit{Temp}$ and $\mathit{Price} < (\mathit{Cost} + Z)$, and for readability, we write $3 \leq X \wedge X \leq 10$ as $3 \leq X \leq 10$. Since constraints limit the acceptable range of parameters within instantiated plan steps, when determining plan compliance with current norms, satisfiability must be checked. We use existing constraint satisfaction techniques [17] to implement a *satisfy* predicate that holds if a given conjunction of constraints admits a solution (if each variable of the constraints admits at least one value that simultaneously fulfils all constraints).

Definition 12 (Satisfy Relation). *satisfy*$(\gamma_0 \wedge \cdots \wedge \gamma_n, \sigma)$ *holds iff* $(\gamma_0 \cdot \sigma \wedge \cdots \wedge \gamma_n \cdot \sigma)$ *is true for some σ.* □

Constraints are associated with first-order predicates, imposing restrictions on their variables. We represent this association as $\varphi \circ \Gamma$, as in, for instance, $\mathit{move}(b_1, X, Y) \circ (100 \leq X \leq 500 \wedge 5 \leq Y \leq 45)$. Now, to define for the core aspect of norms, we use atomic deontic formulae annotated with such constraints.

Definition 13 (Deontic Formula). *An annotated deontic formula ν is any construct of the form $\mathsf{O}_\alpha \varphi \circ \Gamma$ (an obligation) or $\mathsf{F}_\alpha \varphi \circ \Gamma$ (a prohibition), where α is a term, and $\varphi \circ \Gamma$ is a first-order atomic formula φ with constraints Γ.* □

Term α identifies the agent(s) to which the norm is applicable and ρ is the role of such agent(s). $\mathsf{O}_{\alpha:\rho} \varphi \circ (\gamma_1 \wedge \ldots \wedge \gamma_n)$ thus represents an obligation on agent α

taking up role ρ to bring about φ, subject to *all* constraints γ_i, $0 \leq i \leq n$. The γ_i terms express constraints on variables of φ. The relation between constraints and the deontic formula is akin to the *quantifier restrictions* introduced in [7]. If we assume a universal quantification in our annotated deontic formulae, that is, $\forall\alpha.\forall\rho.\forall\boldsymbol{x}(\mathsf{X}_{\alpha:\rho}\varphi \circ \Gamma)$ (where $\boldsymbol{x}$ are all variables occurring in φ and Γ, and X is either O or F) then our formula stands for $\forall\alpha.\forall\rho.\forall\boldsymbol{x}(\Gamma \rightarrow \mathsf{X}_{\alpha:\rho}\varphi)$. Alternatively, if we assume an existential quantification, that is, $\exists\alpha.\exists\rho.\exists\boldsymbol{x}(\mathsf{X}_{\alpha:\rho}\varphi \circ \Gamma)$, then the formula stands for $\exists\alpha.\exists\rho.\exists\boldsymbol{x}(\Gamma \wedge \mathsf{X}_{\alpha:\rho}\varphi)$.

Thus, norms are defined as follows.

Definition 14 (Abstract Norm). *An abstract norm ω^A is a tuple $\langle \nu, Act, Exp, id\rangle$ where: ν is an annotated deontic formula (cf. Definition 13); Act, the* activation condition*, is a conjunction of possibly negated first-order atomic formulae $\varphi_1 \wedge \cdots \wedge \varphi_n$ specifying the condition that must hold in the agent's belief base for the norm to take effect; Exp, the* expiration condition*, is a conjunction of possibly negated first-order atomic formulae $\varphi_1 \wedge \cdots \wedge \varphi_n$ specifying the condition that must hold in the agent's belief base for the norm to stop being in effect; id is a unique norm identifier.* □

We denote a set of abstract norms as Ω^A. If the activation condition of an abstract norm holds, then a *specific norm* is obtained, whereby variables may be instantiated to specific values. Abstract norms generically define circumstances when norms should be adopted and dropped; when norms are adopted, the abstract formulation is instantiated to specific circumstances.

Definition 15 (Specific Norm). *A specific norm ω^S is a tuple $\langle \nu, Act, Exp, \sigma, id\rangle$ where ν, Act, Exp, id are as above and are bound by a substitution σ. We denote a set of specific norms as Ω^S.* □

As agents interact with their environment and with other agents, their perception of reality, as recorded in their sets of beliefs, change. Agents use their beliefs to update their normative positions, adding norms whose activation conditions hold, and removing norms whose expiration conditions holds. Given a set of beliefs *Bel* and a specific norm ω^S of the form $\langle \nu, Act, Exp, \sigma, id\rangle$, then ω^S holds (or is in effect) if, and only if, the following two conditions hold:

1. $Bel \vdash Act \cdot \sigma$; that is, we can deduce $Act \cdot \sigma$ from the set of beliefs, and
2. $Bel \nvdash Exp \cdot \sigma$; that is, we cannot deduce $Exp \cdot \sigma$ from the set of beliefs.

Since beliefs change, norms also change as their activation and expiration conditions may no longer hold, capturing dynamic aspects in our representation.

Example 2. The norms of our scenario are the following abstract norms:
1. $\langle \mathsf{F}_{A:R} evacuate(X, Y) \circ \{Y = W\}, \neg safe(W), safe(W), 1\rangle$
2. $\langle \mathsf{O}_{A:R} reroute(X, Z) \circ \{X + 1 \leq Z \leq X + 3\}, \neg safe(X), safe(X), 2\rangle$

The first norm states that all agents (in all roles) are forbidden to evacuate an area X to an area Y; the prohibition becomes active if area Y (constrained to be W) is unsafe and expires when area Y (constrained to be W) becomes safe;

unifications are dealt with like constraints, hence the need to use a third variable W. The second norm states that all agents (in all roles) are obliged to reroute traffic through Z to avoid area X, but the rerouting must be within nearby zones. The norm becomes active when area X is deemed not safe, and the norm is deactivated when area X becomes safe again. Now, suppose these norms give rise to the following specific norms:
3. $\langle \mathsf{F}_{A:R} evacuate(X, Y) \circ \{Y = W\}, \neg safe(W), safe(W), \{W/3\}, 1\rangle$
4. $\langle \mathsf{F}_{A:R} evacuate(X, Y) \circ \{Y = W\}, \neg safe(W), safe(W), \{W/6\}, 1\rangle$
5. $\langle \mathsf{O}_{A:R} reroute(X, Z) \circ \Gamma, \neg safe(X), safe(X), \{X/2\}, 2\rangle$
That is, abstract Norm 1 gives rise to two specific norms, one instantiating W to 3 and another W to 6. Abstract Norm 2 (shown with constraints abbreviated as Γ to save space) gives rise to one specific norm, instantiating X to 2. ■

For simplicity, we assume an implicit universal quantification over variables in ν, *Act* and *Exp*, but our approach can be extended for any quantification.

4 ν-BDI: a Normative BDI Interpreter

Given the representation of norms as detailed above, we can now address the issues surrounding their integration into an effective BDI architecture. First, we describe the key processes required in the agent interpreter to manage the activation and expiration of norms. Although beliefs are generally [10] assumed to contain exclusively ground first-order predicates, in this paper we store both abstract and specific norms in the belief base. In doing so we avoid adding extra components to the architecture. We extend and adapt the mechanisms to update beliefs and to reason with beliefs, enabling them to deal with norms.

A set of abstract norms is used, together with the current set of beliefs *Bel*, to obtain an updated set of beliefs with all norms with a valid activation condition added, and all norms with a valid expiration condition removed. This is detailed in Algorithm 1: it adds or removes specific norms, preserving abstract norms and ground predicates.

Algorithm 1 uses function $getNorms^A(Bel) = \Omega^A = \{\omega_0^A, \ldots, \omega_n^A\}$ which, given a belief base *Bel* as input, returns the possibly empty set of abstract norms in *Bel*; $getNorms^S(Bel) = \Omega^S$, similarly, returns the specific norms in *Bel*. We also make use of function $getPreds(Bel) = \{\bar{\varphi}_0, \ldots, \bar{\varphi}_m\}$ which returns the ground predicates of a belief base. The algorithm initialises (Lines 2–4) working sets of abstract norms, specific norms and ground predicates, respectively; it also initialises the working belief base Bel', setting it to *Bel*. Lines 5–9 loop through all abstract norms, checking if the ground predicates in the belief base trigger norms' activation conditions. For each norm, the algorithm exhaustively finds possible substitutions σ in which the activation condition holds in *Bel*, and adds the substitution σ to the abstract norm, resulting in the creation of a newly activated, specific norm. Lines 11–15 loop through all specific norms, generating new possible substitutions σ' for which the expiration condition holds, after application of the original substitution σ. This gives rise to a composite

Algorithm 1 Update norms in belief base

```
1: function updateNorms(Bel)
2:     Ω^A ← getNorms^A(Bel)
3:     Φ ← getPreds(Bel)
4:     Bel′ ← Bel
5:     for all ⟨ν, Act, Exp, id⟩ ∈ Ω^A,, do
6:         for all σ such that Φ ⊢ Act · σ do
7:             Bel′ ← Bel′ ∪ {⟨ν, Act, Exp, σ, id⟩}          % add activated norms
8:         end for
9:     end for
10:    Ω^S ← getNorms^S(Bel′)
11:    for all ω^S ∈ Ω^S, ω^S of the form ⟨ν, Act, Exp, σ, id⟩, do
12:        for all σ′ such that Φ ⊢ Exp · σσ′ do
13:            Bel′ ← Bel′ \ {ω^S}                           % remove expired norms
14:        end for
15:    end for
16:    return Bel′
17: end function
```

substitution $\sigma\sigma'$ that ensures the algorithm only removes those specific norms whose expiration conditions hold.

Example 3. Let us suppose we have an abstract norm ω^A:

$$\langle \mathsf{O}_{A:R} use(hlc, X) \circ \Gamma, high\_risk(X), weather(X, poor), 3 \rangle$$

This represents an obligation on all agents/roles to fly a helicopter (represented as *hlc*) over X; the norm becomes active if X is a high-risk area, and the norm expires if the weather conditions in X are poor. The Γ stipulates which areas can be flown over, and its details are not relevant to our example. If we have a belief base $Bel = \{high\_risk(10), \omega^A\}$, where ω^A is the abstract norm above, then we would add to Bel the specific norm

$$\langle \mathsf{O}_{A:R} use(hlc, X) \circ \Gamma, high\_risk(X), weather(X, poor), \{X/10\}, 3 \rangle$$

If, however, the belief base also had a predicate $weather(10, poor)$, then no specific norms would be added, as the expiration condition of the newly added norm would hold – the algorithm adds a specific norm in loop 5–9, then removes it in loop 11–15. ■

4.1 Actions and Norms

As indicated previously, our key concern in this paper is with the impact of norms on plans. Critical to this is determining when an action (represented as an atomic formula φ) is within the scope of influence of a specific norm ω^S. Algorithm 2 defines predicate *inScope* which, given an agent specified by its unique identifier Ag and one of the roles $R \in Rl$ of the agent, holds if a first-order predicate φ is within the influence of a specific norm ω^S (c.f. Definition 15). Line 1 states the format of norm ω^S (where X is either F or O). Line 2 tests if Ag, φ, unify with, respectively, α, φ'; that is, if the agent identifier and the action formula unifies with the corresponding components of a norm. By considering actions

Algorithm 2 Check if action in scope of norm.

1: **function** INSCOPE($Ag, \varphi, \langle \mathsf{X}_\alpha \varphi' \circ \Gamma, Act, Exp, \sigma, id \rangle$)
2: **if** $[(\neg isAction(\varphi) \wedge unify(\langle Ag, \varphi \rangle, \langle \alpha, \varphi' \rangle \cdot \sigma, \sigma')) \vee$
3: $(isAction(\varphi) \wedge$
$\exists \phi \in \varepsilon(\varphi)$ s.t. $unify(\langle Ag, \phi \rangle, \langle \alpha, \varphi' \rangle \cdot \sigma, \sigma'))] \wedge$
4: $satisfy(\Gamma \cdot \sigma, \sigma')$ **then**
5: **return** σ'
6: **else return** $\bot$
7: **end if**
8: **end function**

as having explicit effects in the world-state (c.f. Definition 9), we can check if a norm referring to a world state is affected by the changes introduced by each action. Thus, Line 3 checks that a norm does not unify with the declarative effects $\varepsilon(\varphi)$ of an action φ. The final part of the condition for an action to be in the scope of a norm, in Line 4, checks if the constraints on φ' (instantiated with the substitution σ obtained via *unify*) can be satisfied. This check factors in the substitution σ obtained when the norm was activated (cf. *updateNorms* in Algorithm 1), thus guaranteeing that different versions of the same abstract norm (obtained due to specific values of activation conditions) are adequately handled. If the action is in the scope of a norm, the algorithm terminates in Line 5, returning substitution σ' satisfying the constraints of the specific norm. Conversely, if the conditions of Lines 2–4 are not met, Algorithm 2 terminates with $\bot$ (false) at Line 6.

4.2 Annotating Plans

As indicated in Section 3, one of our primary concerns is with the impact of norms on agent plans in terms of constraints on the values of variables of an action. Since actions and achievable world-states are components of plans, instances of restricted actions and world states must be found and marked with these constraints.

To achieve this, we propose Algorithm 3, which scans a plan, annotating each step within the scope of a norm with constraints stemming from that norm. Each step of the plan is checked against the predicates specified in the specific norms of Ω^S. If a step is within the scope of a norm (Line 9), then the algorithm gradually assembles the constraints of the norms Γ_i, and annotates the plan step with them. If the norm is an obligation, the constraints are added as they appear in the norm (Line 13), refined to the substitutions σ, σ'. If the norm is a prohibition, the constraints are then *negated* (Line 11); formally, $neg((\gamma_1, \ldots, \gamma_n)) = (neg(\gamma_1), \ldots, neg(\gamma_n))$, and each constraint can be negated as $neg(\tau > \tau') = (\tau \leq \tau')$, $neg(\tau < \tau') = (\tau \geq \tau')$, $neg(\tau \geq \tau') = (\tau < \tau')$, and so on. If the step is not in the scope of any norm, then no constraints are added.

Algorithm 3 always terminates as all its loops are over finite constructs and all tests carried out terminate. Its complexity is $|Plib| \times |s| \times |\Omega^S|$; that is, the product of the number of plans, the number of steps of each plan (for simplicity

Algorithm 3 Plan annotation

```
1: function annotatePlans(Ag, Plib, Bel)
2:     Plib' ← ∅ ; Ω^S ← getNorms^S(Bel)
3:     for all plans P ∈ Plib of the form ⟨t, c, [s_1, ..., s_n]⟩ do
4:         Γ' ← ⊤                                          % initialise plan's annotation
5:         for all steps s_i in [s_1, ..., s_n] do
6:             Γ_i ← ⊤                                     % initialise step's annotation
7:             for all ω^S ∈ Ω^S, ω^S of the form ⟨X_α φ ∘ Γ, Act, Exp, σ, id⟩, do
8:                 σ' ← inScope(Ag, s_i, ω^S)
9:                 if σ' ≠ ⊥ then                          % if in scope of norm
10:                    if X = F then
11:                        Γ_i ← Γ_i ∧ (neg(Γ) · σσ')      % negate/add cnstrs.
12:                    else if X = O then
13:                        Γ_i ← Γ_i ∧ (Γ · σσ')           % add constraints.
14:                    end if
15:                end if
16:            end for
17:            Γ' ← Γ' ∧ Γ_i                               % collect annotations
18:        end for
19:        Plib' ← Plib' ∪ {⟨t, c, [(s_1 ∘ Γ_1), ..., (s_n ∘ Γ_n)], Γ'⟩}
20:    end for
21:    return Plib'
22: end function
```

we use the number of steps s of the largest plan), the number of norms, and the number of roles. The algorithm is correct in that it provides a version of the input plan library *Plib* in which every step of each individual plan has been annotated with constraints stemming from norms; if these constraints are satisfiable, the plan can be executed without violating any active norm. Section 4.3 describes how these plan annotations affect the reasoning cycle.

Once plan steps have been annotated, it is possible for an agent to check before executing each step if its execution violates a norm. However, it is inefficient to adopt a plan and execute it partially before discovering that the plan was not, in fact, desirable from the perspective of norm compliance. Fortunately, since the specific values of the variables within a plan are bound when a plan is instantiated, it is possible to determine at plan instantiation if any normative restriction applied to individual plan steps would be violated if the plan is adopted. In order to do this, we must make all annotations available for checking when the plan is instantiated so, at the end of each iteration over the steps of a plan, we collect the annotations into a global plan annotation Γ' (Line 17 of Algorithm 3), which we use later when *selecting* norm compliant plans.

4.3 Selection of Annotated Plans

We have also developed an algorithm as an extension of the traditional plan selection mechanism used in interpreters such as Jason [4], but with the requirement that the plan annotation, customised (via substitution σ) to the instantiation of the plan to event e, be satisfiable. While this new plan selection algorithm enables compliance with norms that change at runtime, it does not give agents the option of non-compliance. It also selects the first norm-compliant plan, without considering other plans that might not only avoid prohibitions, but also achieve a number of obligations.

Simple compliance with norms as just described does not afford an agent freedom to exploit opportunities when rewards outweigh norm penalties or, more importantly, when a task is *impossible* to accomplish without violating some norms. Plan selection in this form is inadequate; we require a plan selection algorithm that allows norm violation in a controlled manner. In this paper, we take a utilitarian approach to violation: successful plans bring a certain positive utility, as does norm compliance, but norm violation brings some negative utility, or penalty. By defining utility functions for *baseUtility()*, *benefit()* and *cost()*, respectively, we are able to rank the plans according to the utility gained by their execution.

We have thus developed an algorithm for an agent capable of ranking plans by utility gain, taking an additional input, l. It identifies the plan with highest net utility on each possible plan, and selects the highest utility plan. Then it updates the intention set according to the plan.

We can now bring together these various steps to build an agent interpreter that includes the norm processing mechanisms just described. This is shown as Algorithm 4 and operates similarly to the original interpreter, but includes extra processes to update norms based on newly perceived beliefs in Line 5 as well as a modified plan selection algorithm in Line 6 that annotates plans and takes into consideration not only the applicability of plans but also their norm compliance before committing to them as intentions. Algorithm 4 is illustrated in Figure 1, with the new processes shown as shaded boxes.

Algorithm 4 Norm-aware BDI interpreter.

```
1: procedure nuAgentInterpreter(⟨Ag, Ev, Bel, Plib, Int, l⟩)
2:     loop
3:         Ev ← updateEvents(Ev)
4:         Bel ← updateBeliefs(Ev, Bel)
5:         Bel ← updateNorms(Bel)
6:         Int ← selectBestPlan(Ev, Bel, Plib, Int, Ag, l)
7:         Int ← executeIntention(Int, Ev)
8:     end loop
9: end procedure
```

5 Related Work

In this section we review some of the most influential previous efforts at creating autonomous agent architectures that are affected by norms, and contrast these with our proposal. In [19] Kollingbaum proposes a language for the specification of normative concepts together with a programming language for norm-governed reasoning agents. The normative concepts (namely, obligations, prohibitions and permissions) and the programming language are given their operational semantics via the NoA Agent Architecture using the Java programming language to explain the meaning of each construct. This approach addresses practical reasoning agents developed using the proposed language and architecture, and although the approach is practical and has clear advantages such as the

possibility to check for norm conflict and potential inconsistencies, heterogeneous agents cannot be accommodated.

The BOID framework [6] is an often cited approach to creating agents capable of reasoning with the usual BDI mental entities, together with obligations. Unlike more traditional BDI approaches, all mental entities are represented as sets of rules. Implementations of BOID make use of propositional rules, containing a set of antecedents and one conclusion. Rules may conflict, and priorities are assigned to rules in order to resolve these conflicts. Most commonly, each mental attitude is assigned the same priority, with beliefs having a higher priority than desires, intentions and obligations. While conceptually elegant, BOID has not been widely used in practical systems, due to a lack of a robust implementation; computational complexity issues when dealing with descriptive languages; and finally, difficulties for the designer in understanding how complex sets of rules may interact. Furthermore, work on BOID represents addresses only a single part of the problem of building an autonomous agent, namely how to infer certain conclusions from certain inputs, in isolation of any other issues.

Recent efforts on the implementation of agent organisations have resulted in the $\mathcal{M}$OISE$^+$ [15] model, which includes of an organisational modelling language that includes constructs for structural, functional and deontic elements [14]. This is primarily a model for the specification of organisational structures, goals and plans. Although some preliminary work has been done on the implementation of individual agents following this model in $\mathcal{J}$-$\mathcal{M}$OISE$^+$ [15], this work considers a domain-specific individual agent as largely transforming organisational events into agent plan invocations while using special actions to affect the organisation as a whole. It is important to point out that the only deontic modalities handled in $\mathcal{M}$OISE$^+$ are those that drive an agent to certain behaviours, whereas our work defines generic language-independent algorithms for dealing with obligations and prohibitions.

More recent research reported in [2] proposes an extension to the BDI programming language 2APL [9], enabling the representation of normative concepts with associated mechanisms for agents to process these. The work addresses aspects our research did not consider (*e.g.*, norms with deadlines and explicit sanctions/rewards); however, we note that our proposal *extends* a reference architecture, rather than advocating a particular technology (namely, 2APL, a specific programming language). Another approach by [8] proposes MaNEA (Magentix2 Norm-Enforcing Architecture), a distributed architecture based on the Magentix2 platform [24], enabling agents to handle uncertainty (of facts about the world), and the salience (*i.e.*, important for a given context) and relevance (*i.e.*, the norm does not apply to a given context) of norms when deciding to comply or not with norms. The proposal is very expressive, but designers would face a challenge to properly model probabilities of beliefs, cognitive states, salience and relevance. Additionally, the approach is aimed only at agents developed in Magentix2.

6 Conclusions

We have described a new norm representation formalism, using constraints as means to *precisely* specify the target of normative stipulations. These constraints are used to determine specific plan instantiations that comply with active norms, effectively determining the acceptable domains for operation. Based on this, we have specified reasoning mechanisms that enable these plan instantiations to restrict behaviour in support of compliance, avoiding violating plans. Importantly, our work enables selective and incremental norm violation in a *controlled* manner in cases where goal achievement would not otherwise be possible, or where norms are explicitly chosen to be ignored.

We have implemented these algorithms within ν-BDI, extending a traditional BDI interpreter. However, our algorithms are sufficiently generic to enable inclusion in *any* BDI interpreter and sufficiently detailed that implementation is straightforward. In addressing normative reasoning to this level of analysis, we have tackled various technical challenges posed by norm processing, such as the detection of activation and expiration conditions and the management of the norm life cycle between these two conditions, through the management of abstract and specific norms. Finally, we have shown the applicability of the mechanisms developed in an emergency evacuation scenario. In future work, we intend to refine the evacuation scenario as a testbed for our interpreter, and handle norm deadlines as well as normative conflicts.

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. N. Alechina, M. Dastani, and B. Logan. Programming norm-aware agents. In *Procs. 11th Int'l Conf. on Autonomous Agents & Multiagent Systems (AAMAS 2012)*, volume 2, pages 1057–1064, Valencia, Spain, June 2012. IFAAMAS.
3. K. R. Apt. *From Logic Programming to Prolog*. Prentice-Hall, U.K., 1997.
4. R. H. Bordini and J. F. Hübner. BDI Agent Programming in AgentSpeak Using Jason. In *Computational Logic in Multi-Agent Systems, 6th Int. Workshop*, pages 143–164. Springer, 2006.
5. R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley, 2007.
6. J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. van der Torre. The boid architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 9–16, 2001.
7. H. Bürckert. A resolution principle for constrained logics. *Artificial Intelligence*, (66):235–271, 1994.
8. N. Criado, E. Argente, P. Noriega, and V. J. Botti. A distributed architecture for enforcing norms in open MAS. In *Advanced Agent Technology*, volume 7068 of *LNCS*, pages 457–471. Springer, 2012.

9. M. Dastani. 2APl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, June 2008.
10. M. d'Inverno, M. Luck, M. Georgeff, D. Kinny, and M. Wooldridge. The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System. *Autonomous Agents and Multi-Agent Systems*, 9(1 - 2):5–53, July 2004.
11. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 1990.
12. A. García-Camino, J.-A. Rodríguez-Aguilar, C. Sierra, and W. W. Vasconcelos. Constraint Rule-Based Programming of Norms for Electronic Institutions. *Journal of Autonomous Agents & Multiagent Systems*, 18(1):186–217, Feb. 2009.
13. D. Grossi, H. Aldewereld, and F. Dignum. Ubi lex, ibi poena: Designing norm enforcement in e-institutions. In *COIN II*, volume 4386 of *LNCS*, pages 101–114. Springer, 2007.
14. J. Hübner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 05 2010.
15. J. F. Hübner, J. S. Sichman, and O. Boissier. Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *Int. J. Agent-Oriented Softw. Eng.*, 1(3/4):370–395, 2007.
16. F. F. Ingrand, R. Chatila, R. Alami, and F. Robert. PRS: A high level supervision and control language for autonomous mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 43–49, Minneapolis, USA, 1996.
17. J. Jaffar, M. J. Maher, K. Marriott, and P. J. Stuckey. The Semantics of Constraint Logic Programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.
18. A. J. I. Jones and M. Sergot. *Deontic Logic in Computer Science: Normative System Specification*, chapter Chapter 12: On the characterisation of law and computer systems: the normative systems perspective, pages 275–307. Wiley Professional Computing Series. Wiley, 1993.
19. M. J. Kollingbaum and T. J. Norman. Norm adoption in the NoA agent architecture. In *Proc. 2nd Int. Joint Conf. on Autonomous Agents & Multi-Agent Systems*, 2003.
20. A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.
21. F. Lopez y Lopez, M. Luck, and M. d'Inverno. A normative framework for agent-based systems. In *Proc. 1st Int. Symposium on Normative Multi-Agent Systems*, 2005.
22. F. Meneguzzi and M. Luck. Norm-based behaviour modification in BDI agents. In *Proc. 8th Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 177–184, 2009.
23. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proc. 2nd Int. Conf. on Principles of Knowledge Representation & Reasoning*, pages 473–484, 1991.
24. J. M. Such, A. García-Fornes, A. Espinosa, and J. Bellver. Magentix2: A privacy-enhancing agent platform. *Engineering Applications of Artificial Intelligence*, 2012. Available on-line 9 July 2012.
25. W. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman. Normative conflict resolution in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2009.
26. G. H. von Wright. *An Essay in Deontic Logic and the General Theory of Action*. North-Holland Publishing Company, 1968.