BDI Reasoning with Normative Considerations

Felipe Meneguzzi[a], Odinaldo Rodrigues[b], Nir Oren[c], Wamberto W. Vasconcelos[c], Michael Luck[b]

[a] *Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul*
Porto Alegre, RS, Brazil
felipe.meneguzzi@pucrs.br
[b] *Department of Informatics, King's College London*
London WC2R 2LS, UK
odinaldo.rodrigues@kcl.ac.uk
michael.luck@kcl.ac.uk
[c] *Department of Computing Science, University of Aberdeen*
Aberdeen AB24 3UE, UK
n.oren@abdn.ac.uk
w.w.vasconcelos@abdn.ac.uk

Abstract

Systems of autonomous and self-interested agents interacting to achieve individual and collective goals may exhibit undesirable or unexpected behaviour if left unconstrained. Norms have been widely proposed as a means of defining and enforcing societal constraints by using the deontic concepts of obligations, permissions and prohibitions to describe what must, may and should not be done, respectively. However, recent efforts to provide norm-enabled agent architectures that guide plan choices suffer from interfering with an agent's reasoning process, and thus limit the agent's autonomy more than is required by the norms alone. In this paper we describe an extension of the Beliefs-Desires-Intentions (BDI) architecture that enables normative reasoning used to help agents choose and customise plans taking norms into account. The paper makes three significant contributions: we provide a formal framework to represent norms compactly and to manage them; we present a formal characterisation of the normative positions induced by norms of an agent's execution within a given time period; and finally, we put forth a mechanism for plan selection and ranking taking into consideration a set of normative restrictions.

Keywords: Multi-agent Systems, Norms, BDI, Constraints, Planning

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. One way to address this issue, in both human and artificial societies, has been the use of norms, which have been proposed as a means of defining and enforcing constraints aimed at ensuring that such undesired behaviour is avoided if agents are norm-compliant [21, 36]. Norms are generally specified using the deontic concepts of obligation, permission and prohibition 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 [61], but more recent work has addressed how norms may be more suitably represented in computational systems [40, 41], their enforcement [28] and their impact on the society as a whole. These abstract away mechanisms through which individual agents reason with and about norms and how individual behaviour is affected by norms [1, 21, 36].

However, practical normative systems require the analysis and specification of the processes through which norms are recognised, decisions about whether to comply with them, and a subsequent adjustment of behaviour. Some recent efforts in this direction have sought to provide norm-enabled architectures [38, 43] to specify how an agent's behaviour may be constrained 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 movement 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 by preventing parts of a plan library from being adopted [43], or replacing the goals of an agent

with the fulfilment of specific norms [38], they suffer from interfering with an agent's reasoning process, and thus limit its autonomy more than is required by the norms alone.

In response, this paper introduces ν-BDI,[1] an extension of the BDI architecture [53] 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. First, we avoid the rather coarse blanket retraction of specific plans (as adopted in previous work) by using *constraints* in our norm representation, thus enabling fine-grained tailoring of plan restrictions to provide efficient plan/norm processing. Second, we provide an efficient plan analysis mechanism used in identifying (and potentially avoiding) plans that violate norms by examining norm scope (in relation to actions), and limiting possible plan instantiations. Finally, we present a technique for the selective and incremental violation of norms in cases where goal achievement would not otherwise be possible. Importantly, unlike some earlier efforts, agents adopting our proposed mechanisms are able to comply with specific normative stipulations with minimum disruption to traditional non-norm influenced reasoning, and so they may decide to violate norms on the basis of a higher utility. In addition to presenting these mechanisms, we provide proofs about important properties of norm fulfilment and violation in the resulting agent behaviour.

In all these aspects, our norm processing algorithms have very low computational cost relative to the reasoning process; they affect specific plan instantiations so as not to over-constrain an agent; and they allow an agent to maintain its autonomy by deciding for itself which norms to follow, based on measures of utility. Moreover, all of these contributions are integrated into a practical agent architecture that is used to provide an empirical evaluation of the algorithms within a running system.

We start the paper with an illustrative example in Section 2 followed by an introduction to the basic BDI agent reasoning model in Section 3. The language for the specification of normative restrictions is then presented in Section 4. The basic reasoning model is then extended to deal with the normative restrictions and used in the plan selection process in Section 5. We then compare our framework to other related work in Section 6 and draw some conclusions in Section 7. Algorithms detailing the implementation of the augmented reasoner are presented in AppendixB.

2. Illustrative Scenario

To illustrate our approach and motivate the norm representation specified in Section 4, we consider a scenario in which software agents support humans responding to an emergency. Here, humans communicate with each other and synchronise their activities through personal assistants responsible for intermediating communication among members of a team, prompting their human counterparts for actions to be carried out, as well as providing information to help humans decide which course of action to take. In our scenario, heavy and continuous rain in an area prone to flooding has put emergency services on alert: a team of humans supported by personal assistants is to carry out alternative plans, depending on the current conditions (*e.g.*, the severity of the flooding, the size of the affected area, buildings more at risk, the people affected, etc). The personal assistants monitor the latest information on weather and levels of water, and also have access to data about high-security installations (*e.g.*, power plants, fuel and chemical depots, etc), high-risk buildings (*e.g.*, hospitals with intensive care patients, primary schools, prisons, etc), routes for evacuation, and so on. We assume the following is one of the available plans:

> "If one detects that the level of flooding in an area X is *medium*, and the area is of *high-risk* (that is, if X contains high-risk buildings), then the plan is: *i)* to isolate the area (*i.e.*, preventing people from entering it); *ii)* to evacuate everyone from X to another area; and *iii)* to reroute traffic away from X."

In our scenario we also assume the two following norms:

1. It is forbidden to evacuate people into an area Y if Y is currently unsafe. This prohibition is revoked once area Y becomes safe again.

2. It is obligatory to deploy emergency services to an area X of high risk from a nearby area Z. This obligation is revoked once area X becomes low risk again.

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

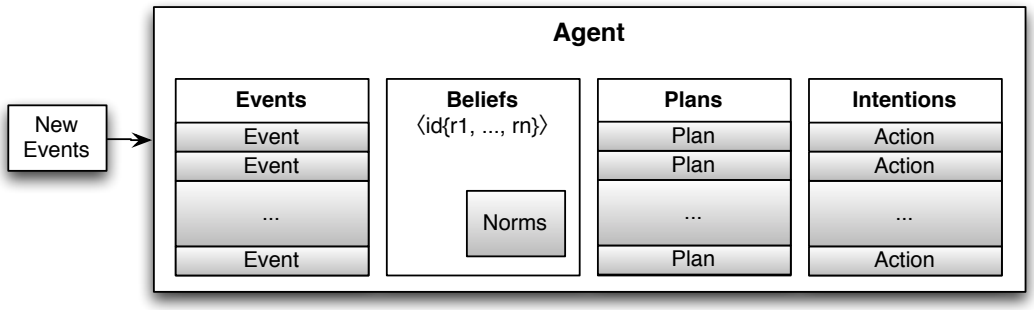

Figure 1: Overview of the architecture of ν-BDI.

These restrictions on the actions affected by the norms above can be considered as being active or inactive depending on the state of the areas involved.

3. Classical BDI Reasoning

In this paper, we take as the basis of our agent interpreter the popular agent model based on the mental attitudes of *belief*, *desire* and *intention* (BDI) [9]. This model sees autonomous agents as having a set of beliefs representing the agent's model of the world, a set of desires representing the agent's potential objectives, and a set of intentions representing the commitment to certain courses of action in order to achieve particular desires. At a high level, reasoning within a BDI agent consists of the following steps. First, an agent perceives the environment and uses these perceptions to update its beliefs. Using the updated beliefs, the agent can then compute which desires are actually possible to accomplish. From these, an agent uses some valuation function to select those desires it wishes to accomplish by committing to plans of actions. We adopt this model due to its popularity in the implementation of practical agent programming languages and interpreters, such as PRS [34], Jack [12], 3APL [16] and Jason [8] amongst others. This ensures that our work has the widest applicability and that, because the model is so well understood gives it wider value even when not directly implementable.

We consider an abstract BDI interpreter inspired by the dMARS architecture [18] and work on AgentSpeak(L) [53]. Figure 1 gives an overview of an augmented BDI agent. The differences from a traditional BDI agent will be introduced in the rest of the paper. To make this paper self-contained, we present preliminary definitions which underpin our mechanisms in AppendixA – we make use of first-order logic constructs, and basic concepts of logic programming such as terms, variables, (negated) literals, substitutions, logical deduction $\vdash$ and semantic entailment $\models$, unification, and most general unifiers [3, 19, 47].

We define an agent in terms of its information model as follows.

Definition 1 (Agent). *An agent is a tuple $\langle Ag, Ev, Bel, Plib, Int\rangle$, where Ag is an agent identifier, Ev is a queue of events, Bel is a belief base consisting of a set of ground literals, $Plib$ is a plan library, and Int is an intention structure.* □

Each of the components of an agent tuple are described in detail below. We start with the event queue:

Definition 2 (Event queue). *An event queue Ev is a sequence of events $[e_1, \ldots, e_n]$ ordered by their occurrence time. Each event e_i records one of following four possibilities:*

i) a belief addition $+\bar{L}$, whereby the literal $\bar{L}$ is to be added to the belief base;

ii) a belief deletion $-\bar{L}$, whereby the literal $\bar{L}$ is to be removed from the belief base;

iii) an achievement-goal addition $+!\bar{L}$, whereby the goal to achieve $\bar{L}$ is selected by the agent;

iv) an achievement-goal deletion $-!\bar{L}$, whereby the goal to achieve $\bar{L}$ has been dropped by the agent; or

v) a test-goal addition $+?\bar{L}$, whereby belief $\bar{L}$ is to be tested by the agent. □

Belief additions are associated with literals perceived as true, and belief deletions with literals no longer perceived as true. Achievement-goal additions and deletions indicate posting of requests to execute a plan. In the context of BDI, an event forms the *invocation condition* of a plan, which is further explained in Definition 3.

The plan library $Plib$ contains a finite and possibly empty set of uninstantiated plans $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ available to the agent. Plans are defined below.

Definition 3 (Plans). *A plan $\mathcal{P}$ is a tuple $\langle t, c, bd\rangle$ where*

- *t is an* invocation condition *that triggers the adoption of the plan, i.e., an event of the form $+\bar{L}$, $-\bar{L}$, $+!\bar{L}$ or $-!\bar{L}$ (cf. Definition 2);*
- *c is a formula representing the* context condition *of the plan; and*
- *bd is the body of the plan constituted of a finite and possibly empty sequence of steps $[s_1, \ldots, s_n]$, where each s_i is either the invocation condition of a plan, or an action identifier (cf. Definition 4).* □

Actions are identified by positive literals, and in order to deal with declarative world-states we must specify what an action entails, so that we are able to use action execution as the target of normative stipulations in Section 4. Thus, we specify the effects of the execution of an action in Definition 4, and provide an algorithmic definition of the subjective meaning of an action with regard to an agent's belief base in Algorithm 8 (in AppendixB). The action model we consider in this paper assumes that actions are always successful. We make this assumption based on the fact that existing BDI architectures do not have a probabilistic model of action failure with defined unexpected action effects. Consequently, our agents consider the expectation of the violation or fulfilment of the norms based only on the intended effects of the actions.

We note that most BDI interpreters do not define an action execution semantics as part of their formal model. This is primarily due to BDI interpreters being considered separately from the environment in which they operate, so the traditional assumption is that an action is sent to the environment, and some effect (presumably known to the agent modeller) is generated as a result. Since we are concerned about defining an agent's reasoning about the future consequences of its actions regarding its compliance to norms, we assume an agent is explicitly aware of the preconditions and effects of the actions included in its plans. Thus, our formalisation includes an explicit action model, much like the one used in recent work on agents with planning capabilities [56, 44].

Before we can formally define an action, we need to informally introduce our assumed execution model. In Situation Calculus [54], an action can be generally defined as a set of axioms. We illustrate this with the action $evacuate(x, y)$, whose objective is to remove everyone from an area x to an area y:

$$\forall z \forall x \forall y \forall s. \begin{pmatrix} person(z,s) & \wedge \\ at(z,x,s) & \wedge \\ high\_risk(x,s) & \wedge \\ \neg high\_risk(y,s) & \end{pmatrix} \rightarrow \begin{pmatrix} at(z,y,do(evacuate(x,y),s)) & \wedge \\ \neg at(z,x,do(evacuate(x,y),s)) & \end{pmatrix} \quad (1)$$

The universal quantification specifies all correct transitions from all states s that satisfy the left-hand side of the implication to state $s' = do(evacuate(x, y), s)$. In classical logic, the axiom can be used as many times as necessary to obtain all of the effects of executing $evacuate(x, y)$ on s. This includes all values of z for which the predicates $person(z, s)$ and $at(z, x, s)$ hold true. However, a planner will in general consider only one such instantiation at a time for some fixed values of x, y and z, and this instantiation will not necessarily include all of the (intended) effects of executing the action $evacuate(x, y)$ on s [45]. We can regard the intended *operational* meaning of an action of this form as

$$\forall x \forall y \forall s. \left(\begin{pmatrix} high\_risk(x,s) \wedge \\ \neg high\_risk(y,s) \end{pmatrix} \rightarrow \forall z \left(\begin{pmatrix} person(z,s) \wedge \\ at(z,x,s) \end{pmatrix} \rightarrow \neg at(z,x,do(evacuate(x,y),s)) \right) \right) \quad (2)$$

Formula 2 therefore contains two sets of conditions. The terms $high\_risk(x, s)$ and $\neg high\_risk(y, s)$ indicate the pre-conditions required to execute $evacuate(x, y)$, i.e., evacuating from x to y requires x to be a high risk area and y

not to be high risk. The second set of conditions, which we call the set of *scope conditions* of the action, specifies the scope of the effects of executing such an action; i.e., for some fixed values of x and y, the effect of evacuating from x to y is to relocate *all* persons currently at location x to location y.

Although (1) and (2) are logically equivalent, *operationally* they have different meanings because at execution time a planner will choose only one amongst all possible instantiation for all of the variables in the context condition of a plan. Therefore, we want to separate the conditions for execution from those that ensure the completeness of the application of the effect. We place the first type in the plan's context condition and leave the variable of the second one to be instantiated at the time the action is executed assuming that at that time **all** possible such instantiations will be effected. In other words, one can think of pre-conditions as required conditions for a unique transition from state s to state $do(evacuate(a, b), s)$ through the specific instantiation $\sigma = \{x/a, y/b\}$, whereas the scope conditions $person(z, s)$ and $at(z, a, s)$ specify all fluents that are actually affected by this transition. This is formally defined below.

Definition 4 (Action). *An action α is a tuple $\langle \varphi, \varpi, \varepsilon^+, \varepsilon^- \rangle$, where*

- *$\varphi = p_i^n(\tau_1, \ldots, \tau_n)$ is the action α's identifier, where the terms τ_i are either constants or variables and contain all variables appearing in ϖ, ε^+ and ε^- (in some pre-defined order);*
- *ϖ is a first-order formula representing α's scope conditions;*
- *ε^+ is a set of literals to be added to the belief base, representing α's positive effects; and*
- *ε^- is a set of literals to be removed from the belief base, representing α's negative effects.*
- *all variables in $vars(\varepsilon^-) \cup vars(\varepsilon^+)$ appear either in ϖ or in the invocation or context conditions of the plan in which α appears.* □

We assume that an action α's pre-conditions are taken care of by the context condition of the plan $\mathcal{P}$. At the point of the action α's execution some substitution σ_α may need to be applied as the result of an earlier instantiation of variables occurring in the plan. This substitution needs to be subsequently applied throughout α's scope condition and effects.

We assume that $\varepsilon^+ \cap \varepsilon^- = \varnothing$, for all actions, so that the order of the application of their effects is irrelevant. Finally, we refer to the set of all possible actions as $Actions$. The effects of the execution of an action on a belief base can be defined in any sensible way. For simplicity, we assume an action is applied as follows.

Definition 5 (Belief base update through action execution). *Let Bel be a belief base, $\alpha = \langle \varphi, \varpi, \varepsilon^+, \varepsilon^- \rangle$ an action, and σ the substitution used in the plan of which α is part of. The belief base obtained from Bel through the execution of α with substitution σ, $do(\alpha, \sigma, Bel)$, is defined as:*

$$do(\alpha, \sigma, Bel) = \big(Bel \backslash Del(\alpha, \sigma, Bel)\big) \cup Add(\alpha, \sigma, Bel)$$

where

$$Del(\alpha, \sigma, Bel) = \{(L \cdot \sigma \cdot \sigma') \mid (Bel \vdash \varpi \cdot \sigma \cdot \sigma') \wedge (L \in \varepsilon^-)\}$$
$$Add(\alpha, \sigma, Bel) = \{(L \cdot \sigma \cdot \sigma') \mid (Bel \vdash \varpi \cdot \sigma \cdot \sigma') \wedge (L \in \varepsilon^+)\}$$

for all substitutions σ'.

That is, an action is applied with some substitution σ originating from the plan's activation and context conditions. For all substitutions σ' such that $Bel \vdash \varpi \cdot \sigma \cdot \sigma'$ holds, all literals in ε^- instantiated with $\sigma \cdot \sigma'$ are removed from Bel and all literals in ε^+ instantiated with $\sigma \cdot \sigma'$ are then added to it.

This ensures that the effect of executing any action on a belief base (and consequently any plan) is well defined. Obviously, a plan may be itself ill-defined, in the sense that some of its actions can never have any effect (*e.g.*, if an action's post-conditions negate the pre-conditions of a subsequent action in the plan); or that by the time some of its actions execute, some of these actions' pre-conditions no longer hold (*e.g.*, if concurrent plan execution is allowed). However, we do not consider this further since it is outside the scope of this paper.

If an action's pre-conditions depend on the post-conditions of the action appearing earlier in a plan, these conditions cannot be checked at the time the action is executed. Putting the pre-conditions in the context of the plan is of no help, because the context is checked when the plan is considered for execution and the conditions depend on the result of the executions taking place afterwards. A modeller has two options in this case: to assume that the action will successfully execute and hence assume that by the time the action is executed the pre-conditions will be satisfied; or to put the action in an embedded plan to be recursively invoked and whose context includes the action's pre-conditions. This will ensure that the plan is invoked only if the pre-conditions of the action are satisfied. Remedial plans can also be specified if these conditions cannot be satisfied.

Now, given the restrictions we have placed on actions, we can proceed to prove the first property we require to show an agent's ability to reason about norms in the future.

Proposition 1 (Ground Effects). *Let σ be the substitution with which an action α is applied to a belief base Bel as part of a plan $\mathcal{P}$'s execution. The sets $Del(\alpha, \sigma, Bel)$ and $Add(\alpha, \sigma, Bel)$ only contain ground literals.*

Proof 1. *According to Definition 1, an agent's initial belief base contains only ground literals. The plan $\mathcal{P}$ can only be applied to a belief base Bel with substitution σ if Bel satisfies the instantiation of $\mathcal{P}$'s context and invocation conditions with σ. Substitution σ will therefore instantiate all variables in these conditions.*

Furthermore, belief additions coming from action executions are also ground, since by Definition 5, if $Bel \vdash \varpi \cdot \sigma \cdot \sigma'$, then $\varpi \cdot \sigma \cdot \sigma'$ is ground. Therefore, $\sigma \cdot \sigma'$ instantiates every variable in $vars(\varpi)$ with a constant. By Definition 4, all variables in $vars(\varepsilon^-) \cup vars(\varepsilon^+)$ appear either in $\mathcal{P}$'s conditions or in α's scope condition; therefore according to Definition 5, $\sigma \cdot \sigma'$ will also instantiate every variable in $Del(\alpha, \sigma, Bel)$ and $Add(\alpha, \sigma, Bel)$ with a constant.

Within an agent's plans, action invocations and belief modifications are both represented in terms of predicate symbols.[2] However, within the body of a plan, belief predicates only appear associated with the symbols for addition and deletion (cf. Definition 3), denoting updates to the belief base. Conversely, predicates referring to actions (*i.e.* their identifier, cf. Definition 4) appear on their own within the body of a plan, denoting that an action is to be executed. Thus, a step φ within a plan denotes the execution of an action identified by predicate φ, whereas a step $+\varphi$ represents the addition of the belief φ to the agent's belief base. We illustrate a complete plan in Example 1, noting that for simplicity, this plan contains only action identifiers. Note that we assume all variables in a plan to be *universally* quantified. To simplify the notation, we omit the universal quantifier and use capital letters to represent implicitly quantified variables, following Prolog's notation [3].

Example 1. *The plan $\mathcal{P}$ we consider in our scenario is represented as follows:*

$$\mathcal{P} = \left\langle +level(X, medium), \left(\begin{array}{c} high\_risk(X) \wedge \neg high\_risk(Y) \wedge \\ \neg high\_risk(Z) \wedge (Y < Z) \end{array} \right), \left[\begin{array}{c} isolate(X), \\ evacuate(X, Y), \\ deploy(Z, X) \end{array} \right] \right\rangle$$

That is, the plan is invoked by the addition of the belief $level(X, medium)$ to the belief base, stating that the level of emergency of area X is $medium$. The context of the activation contains conditions on the plan itself and also the pre-conditions of the actions: that is, X is a high risk area and Y and Z are distinct non-high risk areas, with Z greater than Y. In this context, the plan is:

i) to $isolate$ the area X (thus preventing anyone from entering it)
identifier: $isolate(X)$; **pre-conditions:** *none;* **scope:** $\top$; $\varepsilon^- = \varnothing$; $\varepsilon^+ = \{blocked(X), \neg safe(X+1)\}$

ii) to $evacuate$ all persons P in X, moving them to a non-high risk area Y
identifier: $evacuate(X, Y)$; **pre-conditions:** $high\_risk(X) \wedge \neg high\_risk(Y)$; **scope:** $person(P) \wedge at(P, X)$; $\varepsilon^- = \{at(P, X)\}$; $\varepsilon^+ = \{at(P, Y), clear(X)\}$

and

[2] We note that, as beliefs are literals, they may also be negated.

iii) to deploy emergency services from a non-risk area Z to X
identifier: $deploy(Z, X)$; **pre-conditions:** $high\_risk(X) \wedge clear(X) \wedge \neg high\_risk(Z)$;
scope: $emergency\_person(P) \wedge at(P, Z)$;
$\varepsilon^- = \{at(P, Z), \neg safe(X + 1)\}$; $\varepsilon^+ = \{at(P, X), secured(X), safe(X + 1)\}$

We illustrate the effects of the action $evacuate(X, Y)$ within this plan. Suppose the event $+level(1, medium)$ triggers the activation condition of the plan and that the agent's belief base is the base Bel_0 described below.

$$Bel_0 = \left\{ \begin{array}{l} high\_risk(1), \neg high\_risk(2), \neg high\_risk(3), \neg high\_risk(4), \neg high\_risk(5), \\ person(john), person(mary), person(susan), \\ at(john, 1), at(mary, 1), at(susan, 2), \\ emergency\_person(lee), at(lee, 3) \end{array} \right\}$$

Several instantiations satisfy the context condition of the plan. Consider instantiation $\sigma = \{X/1, Y/2, Z/3\}$, which will yield the sequence of actions $isolate(1), evacuate(1, 2), deploy(3, 1)$. The scope condition of $isolate(X)$ is trivially satisfied. After it is executed, the belief base becomes $Bel_1 = do(isolate(X), \{X/1, Y/2, Z/3\}, Bel) = Bel \cup \{blocked(1), \neg safe(2)\}$. The next action in the plan is $evacuate(X, Y)$, whose scope is

$$person(P) \wedge at(P, X)$$

Notice that $Bel_1 \vdash (person(P) \wedge at(P, X)) \cdot \{X/1, Y/2, Z/3\} \cdot \{P/john\}$ and $Bel_1 \vdash (person(P) \wedge at(P, 1)) \cdot \{X/1, Y/2, Z/3\} \cdot \{P/mary\}$. Therefore, according to Definition 5, $Del(evacuate(X, Y), \sigma, Bel) = \{at(john, 1), at(mary, 1)\}$ and $Add(evacuate(X, Y), \sigma, Bel) = \{at(john, 2), at(mary, 2), clear(1)\}$ and hence $do(evacuate(X, Y), \sigma, Bel_1)$ is the belief base Bel_2 below.

$$Bel_2 = \left\{ \begin{array}{l} high\_risk(1), \neg high\_risk(2), \neg high\_risk(3), \neg high\_risk(4), \\ person(john), person(mary), person(susan), \\ emergency\_person(lee), at(lee, 3), \\ blocked(1), \neg safe(2), \\ at(susan, 2), at(john, 2), at(mary, 2), clear(1) \end{array} \right\}$$

The last action in the plan, $deploy(3, 1)$, would execute in a similar fashion. ■

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

Definition 6 (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 plan execution of a BDI agent. This model is used with little or no modification in various agent interpreters such as PRS [33], Jason [7] and others. However, these interpreters have no mechanism to process social norms, a mechanism that is increasingly being used to implement agent coordination without the need for explicit communication [36]. In order to reason about norms, such interpreters must rely on a designer to hard-code any norm-influenced behaviour. As discussed in the introduction, our goal is to increase a BDI agent's capability to operate in a heterogeneous by making them norm-aware. Thus, in the following sections we describe the normative framework within which ν-BDI operates and the algorithms necessary for generic normative behaviour. We assume the reader is familiar with the standard BDI reasoning cycle and do not discuss it in detail, but include a summary of the algorithms (adapted from [44]) required for the reasoning cycle in AppendixB.1. In what follows we elaborate on these algorithms to build our normative reasoning mechanism.

Definition 7 (Relevant and applicable plans). *Let $\mathcal{P} = \langle t, c, bd \rangle$ be a plan and e an event. We say $\mathcal{P}$* is relevant *to the event e with substitution σ_t if $unify(t, e, \sigma_t)$ holds for some substitution σ_t. We say that a plan $\mathcal{P} = \langle t, c, bd \rangle$ relevant to an event e with substitution σ_t* is applicable to a belief base Bel with substitution σ in response to the event e, *if there exists a substitution σ_c such that $Bel \vdash c \cdot \sigma_t \cdot \sigma_c$. The set of all pairs $\langle \mathcal{P}, \sigma \rangle$ of plans and substitutions $\sigma = \sigma_t \cdot \sigma_c$ applicable to a belief base Bel in reaction to an event e will be denoted by $plan\_options(e, Bel)$.*

We assume the existence of a procedure $select\_plan(e, Bel) = \langle \mathcal{P}, \sigma \rangle$ to select one plan $\mathcal{P}$ applicable to belief base Bel with substitution σ as a reaction to the event e amongst all options in $plan\_options(e, Bel)$. If there are no options, $select\_plan(e, Bel)$ *fails*. In what follows $last(Seq)$ returns the last element in the sequence Seq.

Definition 8 (Plan Application). *Let $\mathcal{P} = \langle t, c, bd \rangle$ be a plan applicable to the belief base Bel with substitution σ. The application of $\mathcal{P}$ to Bel with substitution σ, $apply(\mathcal{P}, \sigma, Bel)$, generates the pair $\langle ASeq, BSeq \rangle$, where $ASeq$ is a sequence of actions and $BSeq$ is a sequence of belief bases defined in terms of the recursive application of $\mathcal{P}$'s steps as follows.*

$$apply(\mathcal{P}, \sigma, Bel) = apply\_steps(bd, \sigma, Bel)$$

where

$$apply\_steps([s], \sigma, Bel) = \begin{cases} \langle [s], [do(s, \sigma, Bel)] \rangle, & \text{if } s \text{ is an action identifier;} \\ apply(\mathcal{P}', \sigma', Bel), & \text{if } s \text{ is the invocation condition of a plan and} \\ & select\_plan(s, Bel) = \langle \mathcal{P}', \sigma' \rangle \text{ for some } \sigma' \\ \langle [nothing], [Bel] \rangle & \text{if } s \text{ is the invocation condition of a plan and} \\ & select\_plan(s, Bel) \text{ fails} \end{cases}$$

$$\begin{aligned} apply\_steps([s_1, \ldots, s_n], \sigma, Bel) = \; & \langle ASeq, BSeq \rangle, \text{where} \\ & apply\_steps([s_1], \sigma, Bel) = \langle ASeq', BSeq' \rangle, \\ & apply\_steps([s_2, \ldots, s_n], \sigma, last(BSeq')) = \langle ASeq'', BSeq'' \rangle, \\ & ASeq = ASeq' :: ASeq'', \\ & BSeq = BSeq' :: BSeq'' \end{aligned}$$

In the definition above, "$nothing$" represents a null action identifier with no effect.

Definition 9 (Actions in a plan). *Let $\mathcal{P}$ be a plan applicable to belief base Bel with σ, and $apply(\mathcal{P}, \sigma, Bel) = \langle ASeq, BSeq \rangle$ the result of this application. The actions in $\mathcal{P}$ relative to Bel and substitution σ, $actions(\mathcal{P}, \sigma, Bel)$, correspond to the sequence $ASeq$.*

Definition 10 (Bases generated by the execution of a plan). *Let $\mathcal{P}$ be a plan applicable to Bel with substitution σ and $apply(\mathcal{P}, \sigma, Bel) = \langle ASeq, BSeq \rangle$, the result of this application. The execution of $\mathcal{P}$ on Bel with σ results in the sequence of belief bases $BSeq$. We denote this by $bases(\mathcal{P}, \sigma, Bel)$.*

The definitions above tell us about the effects of executing a plan on a belief base under *idealised* conditions, *i.e.*, that plans are executed serially and that all of their steps are applied. The procedure for computing these effects will terminate as long as there is not an infinite sequence of embedded plan invocations (as, *e.g.*, in a loop). We also assume (and explicitly enforce via Algorithms 1 and 9) that one action at most is executed per reasoning cycle, so we can identify exactly what caused each change in the belief base.

4. Dealing with Norms

Deontic concepts, namely obligations, permissions and prohibitions, identify what must, may and should not be done in a MAS. These concepts are usually encoded by norms, and specify enforceable societal constraints. Typically, an agent acting as part of a society or organisation will act in a manner consistent with its normative requirements (thus avoiding any penalties incurred for normative violation) but, in exceptional cases, may act contrary to its norms in order to achieve some overarching goal. In this paper, we seek to identify the computational mechanisms required to enable a BDI agent to choose whether to comply with, or violate, its norms. Since norms may affect behaviour, we pay particular attention to the mechanisms that regulate their scope of influence.

We offer two distinct means of addressing this: first, we draw on aspects similar to those presented in [21] and [38] in that our norms are *conditional*, both for activation (when they come into force) and expiration (when they cease effect), limiting their application to periods of time; and second, we add *constraints* [35], limiting their application to particular plans and actions, and ensuring that norms are not over-restrictive. For example, following the scenario of Section 2, if particular parts within the area of operations become unsafe as the rescue operation is underway, we would like to concisely represent prohibitions for rescue workers to move through these unsafe areas. Using

more traditional normative representations, each unsafe area would have to be covered by a particular norm, one per ground instance of the movement action bound to each area. However, in ν-BDI, we can express in a single norm the prohibition affecting multiple ground instances of the movement action using an associated constraint to specify the range of areas that are to be avoided. Regarding the persistence of an obligation past its deadline, we take the stance of our own previous work [46], that an obligation, once expired, vanishes (possibly with penalties applied for any possible violation).

In order to achieve such properties, we formally define the class of norms that our interpreter is able to process in Section 4.1. Using this representation, we proceed to defining the stats in which a norm may be in terms of transitions into an active state and subsequently to either fulfilled or violated states in Section 4.2. Finally, we formally define the meaning of our norms in terms of compliance for each type of norm our representation allows in Section 4.3.

4.1. Norm Representation

The representation we employ in this paper allows for norms to refer to either *actions* or *world states*. Although this might seem redundant at first, since if a norm forbids performance of a particular action with some effects, other actions with the same effects can be executed, this phenomenon exists in the real world. For example, no legislation forbids one to be under the influence of opiate narcotics, but if one achieves this state through the use of illegal drugs (*e.g.*, heroin), there is a violation, whereas if this is achieved through a medically prescribed opiate drug (*e.g.*, morphine) there is no violation. Since norms specify behavioural orientations aimed at inducing agents to act in particular ways in a society, a simplistic view of obligations is that they specify actions that an agent must execute and world states that an agent must bring about, and conversely for prohibitions. Such specification of the actions and world states targeted by norms becomes more complicated when one considers a first-order representation with variables. This forces a designer to either specify blanket norms (which might be too broad), or to create one norm for each possible instantiation of an action or predicate.

Manually creating the complete set of norms is time-consuming and error-prone, and this is an eminently human-led task (possibly with some semi-automated support). In order to address the limitations of specifying norms either as blanket norms or an exhaustive enumeration of possible ground instantiations, we propose a representation that specifies normative stipulations not only in terms of the actions and beliefs to be regulated, but also applies *constraints* on the bindings of the variables used by these actions and beliefs. The norm representation we propose extends and adapts the notation for specifying norms of [60] (which, in turn, extends that of [50], complementing it with *constraints* [35]). Constraints are used to further *refine* the scope of influence of norms on actions, and are defined as follows.

Definition 11 (Constraint). *A* constraint Γ *is the conjunction of one or more constructs* γ *each of the form* $\tau \lhd \tau'$*, where* τ *and* τ' *are terms or arithmetic expressions involving terms and* $\lhd$ *is one of the infix binary operators* $=$*,* $\neq$*,* $>$*,* $\geq$*,* $<$*, or* $\leq$*.* □

We also represent $\gamma_1 \wedge \ldots \wedge \gamma_n$ as a set $\{\gamma_1, \ldots, \gamma_n\}$. The arithmetic operations in a construct will be written in their infix form, as in, *e.g.*, $23 > X$ and $Y < (X + Z)$. To improve readability, constraints of the form $W \leq 30 \wedge W \geq 20$ will be abbreviated to $20 \leq W \leq 30$.

Since constraints limit the values bound to the variables within a plan step to an accepted range of values (as we shall see later), when determining whether a plan complies with existing norms, the consistency of the variable values with the constraints on them must be checked. We define predicate $satisfy$ which holds if, and only if, a constraint is satisfiable.[3] If it is satisfiable, then the solution is represented as a substitution σ to be understood as a combination of variable bindings that satisfy the constraints.

Definition 12 (Satisfy relation). $satisfy(\gamma_0 \wedge \cdots \wedge \gamma_n, \sigma)$ *holds if, and only if,* $(\gamma_0 \cdot \sigma \wedge \cdots \wedge \gamma_n \cdot \sigma)$ *is satisfiable.*□

Constraints operate on formulae imposing restrictions on the variable bindings. We represent this association as $\varphi \circ \Gamma$, where φ is an atomic formula and Γ is a constraint. Thus, in $move(b_1, X, Y) \circ (100 \leq X \leq 500 \wedge 5 \leq Y \leq 45)$, the acceptable values for X are between 100 and 500 and the acceptable values for Y are between 5 and 45.

Our norms consist of a deontic formula possibly annotated with a constraint of the type given in Definition 11. The deontic formula expresses the obligation or prohibition of an agent to execute an action or to hold a particular

[3] This predicate can be implemented in a straightforward fashion, using existing constraint satisfaction techniques [35].

belief. The scope of the obligation or prohibition is further limited by the constraint, if it is present. We call this combination of formula and constraint an *annotated deontic formula*, and consider it formally next.

Definition 13 (Annotated deontic formula). *An* annotated deontic formula ν *is an expression of the form:*

- $\mathsf{O}_A\varphi \circ \Gamma$ *(representing an obligation); or*
- $\mathsf{F}_A\varphi \circ \Gamma$ *(representing a prohibition)*

where A is either a constant or a variable and it identifies the agent(s) to which the norm applies, and $\varphi \circ \Gamma$ is an atomic formula φ with associated constraints Γ. □

If an annotated deontic formula contains a variable as the agent identifier, then this variable is assumed to be universally quantified, i.e., the norm applies to all agents. If it contains a constant, then the norm applies to the agent named by the constant only. As for the variables in $\varphi \circ \Gamma$, they are implicitly *existentially quantified* in obligations and *universally quantified* in prohibitions. Let $\overrightarrow{x}$ denote all variables occurring in φ and Γ. The obligation $\mathsf{O}_A\varphi \circ \Gamma$ stands for $\forall A.\exists \overrightarrow{x}.\mathsf{O}_A\varphi \wedge \Gamma$, *i.e.*, all agents must achieve φ with *at least* one binding of its variables satisfying the constraint Γ. The prohibition $\mathsf{F}_A\varphi \circ \Gamma$ stands for $\forall A.\forall \overrightarrow{x}.\mathsf{F}_A\varphi \wedge \Gamma$, *i.e.*, all agents must refrain from achieving φ for *any* variable binding satisfying Γ.

A third kind of annotated deontic formula would be $\mathsf{P}_A\varphi \circ \Gamma$, representing a *permission*, and which can be defined in terms of an obligation (as usual) as $\mathsf{P}_A\varphi \circ \Gamma =_{\text{def}} \neg\mathsf{O}_A\neg\varphi \circ \Gamma$ [30]. However, for the sake of simplicity, in this work we only consider annotated deontic formulae representing obligations and prohibitions.

Our representation here is precise (as constraints provide a fine-grained way to specify values of variables) and compact (as constrained predicates amount to possibly infinite sets of ground formulae). In what follows, we refer to the same agent A allowing us to drop the agent subscript from the deontic formulae for clarity of exposition. Let us suppose that the unary predicates p, q, and r are associated with action identifiers in the plans $\mathcal{P}_1$–$\mathcal{P}_4$ shown in Table 1, and that the norms $\mathsf{F}p(a)$ and $\mathsf{O}q(b)$ remain active for the duration of the plans. In Table 1 we represent

plan	actions	$\mathsf{F}p(a)$	$\mathsf{O}q(b)$
$\mathcal{P}_1$	$[s(a,b), p(a), q(a), r(a)]$	×	×
$\mathcal{P}_2$	$[q(a), p(b), s(a,b), r(a)]$	✓	×
$\mathcal{P}_3$	$[q(b), p(b), s(a,b), r(a)]$	×	✓
$\mathcal{P}_4$	$[q(b), p(a), s(a,b), r(a)]$	✓	✓

Table 1: Sample plans, their actions and norm compliance

norm compliance as "✓" and norm violation as "×". The execution of plan $\mathcal{P}_1$ violates the prohibition and does not fulfil the obligation. Similarly, even though plan $\mathcal{P}_2$ does not violate the prohibition, it fails to fulfil the obligation. Plan $\mathcal{P}_3$ fulfils the obligation but violates the prohibition, whereas plan $\mathcal{P}_4$ fulfils the obligation without violating the prohibition. If agent A is to act in a norm-compliant manner, it should give priority to $\mathcal{P}_4$ in this case.

In what follows, we formalise norms in such a way as to enable BDI agents to explicitly reason about their compliance with regard to their plan execution.

We introduce norms in two parts: first we define *abstract* norms, which are in the form usually specified in the environment and perceived by agents; then we define *specific* norms, which are instantiated abstract norms in effect for a particular belief base Bel. Agents use the specific version when considering plan alternatives with respect to norm compliance.

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* invocation condition*, is a conjunction of literals $L_1 \wedge \cdots \wedge L_m$ specifying the condition that triggers the potential activation of the norm;*

- *Exp, the* expiration condition*, is a conjunction of literals $L_1 \wedge \cdots \wedge L_n$ specifying the condition that triggers the potential de-activation of the norm;*
- *id is a unique norm identifier.* □

We assume an implicit universal quantification over variables in Act and Exp, in addition to the implicit quantification of the annotated deontic formulae.

Thus, an abstract norm provides a *schema* for activation and deactivation of specific norms. In particular, a specific norm is generated for each combination of variable binding satisfying the activation condition of an abstract norm. The specific norm is dropped when its expiration condition holds thus ensuring that only the correct instances of an abstract norm are de-activated.

Definition 15 (Specific norm). *Let $\omega^A = \langle \nu, Act, Exp, id \rangle$ be an abstract norm and σ some substitution that grounds all of the variables in Act. A* specific version *of ω^A with substitution σ, is a norm of the form $\omega^S = \langle \nu{\cdot}\sigma, Exp{\cdot}\sigma, nid \rangle$, where nid is a unique specific norm identifier.* □

An abstract norm can be activated multiple times, possibly with the same substitution, and thus generate multiple identical specific norms along the reasoning timeline. In order to distinguish these, we associate a unique identifier nid for each activation, which must relate the specific norm back to its generating abstract counterpart. The identifier can then later be used to associate violations and fulfilments not only with each specific instance of the norm but also with its abstract counterpart (which we call "parent"). Since specific norms can only be de-activated, they do not need to keep the activation condition in their parent abstract norm.

Notice that constraints over the activation (and expiration) of norms are relatively commonplace. In order to capture a norm of the form "if $X < 5$ then an obligation exists to ensure that $Y > 7$", we must be able to represent constraints in the activation (and expiration) condition of the norm (as in, *e.g.*, [48]). For simplicity, we assume that such a constraint can be captured as part of the condition's first order formulae, and leave an explicit treatment of this issue to future work.

Example 2. *The norms of our scenario are represented as the following abstract norms:*

1. $\langle \mathsf{F}_A evacuate(X, Y), \neg safe(Y), safe(Y), f1 \rangle$
2. $\langle \mathsf{O}_A deploy(Z, X) \circ \{X + 1 \leq Z \leq X + 3\}, high\_risk(X) \wedge clear(X), secured(X), o1 \rangle$

Norm $f1$ stipulates that all agents are forbidden to evacuate people to an area that is not safe. The norm is activated when an area Y becomes unsafe and expires when Y becomes safe again. Norm $o1$ states that all agents are obliged to deploy emergency services to a high risk area X. The deployment must be from a nearby zone Z. The norm becomes active when the area X is deemed to be high risk and has been evacuated ($clear(X)$), and it is deactivated once the area X is secured.

Now assume that an agent's belief base contains the belief $high\_risk(1)$ and that subsequently, the beliefs $\neg safe(1)$ and $clear(1)$, which were not currently in it, are then added to it. The addition will trigger the activation of the following specific norms.

1. $\langle \mathsf{F}_A evacuate(X, 2), safe(2), f1.1 \rangle$
2. $\langle \mathsf{O}_A deploy(Z, 1) \circ \{2 \leq Z \leq 4\}, secured(1), o1.1 \rangle$

That is, the abstract norm $f1$ gives rise to the specific norm $f1.1$ by instantiating X to 1 and the abstract norm $o1$ gives rise to the specific norm $o1.1$ also by instantiating X to 1. The actual ids of these norms are not important at this stage but they must uniquely identify them as mentioned earlier. Specific norms remain active until their expiration conditions are derived following a belief base update, this being the main reason why we keep the instantiated versions of the expiration conditions in them. Thus, if, say, $safe(2)$ were to be added to the belief base, this would have no effect on the specific norm $f1.1$, allowing the prohibition $\mathsf{F}_A evacuate(X, 3)$ to persist until $safe(3)$ is included. ■

Although traditionally the belief base of an agent only contains ground literals (cf. Definition. 1), in order to simplify our representation we allow it to contain the abstract and specific norms as well (as tuples). We then use $\Omega^A(Bel)$ and $\Omega^S(Bel)$ to denote the sets of abstract and specific norms included in a belief base Bel, respectively.

As agents interact with their environment and with other agents, their perception of reality as recorded in their belief bases changes. Agents use their beliefs to update their normative positions, adding specific norms for the abstract ones whose activation condition holds, and removing specific norms whose expiration condition holds in a similar fashion to the addition and removal of beliefs. We provide a norm updating mechanism for BDI agents in Section 5, with a detailed algorithm. Conceptually speaking, there is no real need for specific norms. One could consider all possible instantiations of the abstract norms that satisfy the norms' activation conditions, but this would have to be done repeatedly for every applicable plan when considering the plans' compliance with the norms. The instantiation of norms avoids this repetition by performing this computation once every time the belief base changes. Once the set of specific norms is computed, it identifies only the norms that need to be checked for compliance for all applicable plans.

The problem of updating the set of specific norms for a new belief base Bel is solved in a straightforward and efficient manner in order to allow it to be used in a *practical* programming environment. Although mechanisms exploring non-monotonic or modal aspects could make the norm update procedure more sophisticated, the complexity of these even for simple logic fragments is very high [42, 59].

4.2. Norm Update

We have seen that abstract norms lead to the generation of specific norms when their invocation condition is inferred from an agent's belief base. Similarly, when a specific norm's expiration condition is inferred, it expires, and no longer needs to be considered by the agent. As long as the set of specific norms is regularly updated, it is sufficient for checking norm compliance. The update process must occur after every belief change and it is formalised as follows.

Definition 16 (Norm activation). *Let Bel be a belief base and $\omega^A = \langle \nu, Act, Exp, id \rangle \in \Omega^A(Bel)$ an abstract norm such that*

- *$Bel \vdash Act \cdot \sigma$ for some substitution σ and $Bel \nvdash Exp \cdot \sigma$; and*
- *$\langle \nu \cdot \sigma, Exp \cdot \sigma, id' \rangle \notin \Omega^S(Bel)$ for any id'*

We say that the specific norm $\omega^S = \langle \nu \cdot \sigma, Exp \cdot \sigma, nid \rangle$, where nid is a new identifier, is activated by Bel. *We denote the set of all specific norms activated in this way by a belief base Bel as $activate(Bel)$.*

That is, if the invocation condition $Act \cdot \sigma$ of an abstract norm ω^A follows from a belief base Bel for some ground substitution σ and its expiration condition Exp instantiated with σ does not follow from Bel and no specific norm of the form $\langle \nu \cdot \sigma, Exp \cdot \sigma, id' \rangle$ is in Bel for some identifier id', then that ω^A is activated by Bel and we construct the specific norm $\omega^S = \langle \nu \cdot \sigma, Exp \cdot \sigma, nid \rangle$ from ω^A, σ and a new identifier nid. Note that it is possible that the activation and expiration conditions of a norm hold simultaneously. In such cases, we believe it is reasonable not to activate the norm and Definition 16 reflects this. It is easy to see that the belief base that activates a norm is the first one to contain it.[4]

Definition 17 (Norm de-activation). *Let Bel be a belief base and $\omega^S = \langle \nu, Exp, id \rangle \in \Omega^S(Bel)$ a specific norm. ω^S* is de-activated by Bel *if $Bel \vdash Exp \cdot \sigma$ for some substitution σ. We denote the set of specific norms de-activated by a belief base Bel by $deactivate(Bel)$.*

We can now consider how to update a set of specific norms with respect to a belief base Bel.

[4] A belief base may satisfy the activation condition of an already active norm but, by definition, only the base that satisfies the activation condition of a not yet active norm is said to activate it.

Definition 18 (Update of specific norms). *Let Bel be a belief base. The update of the set of specific norms in Bel is defined as*

$$\Omega^S(Bel) = \big(\Omega^S(Bel) \cup activate(Bel)\big) \backslash deactivate(Bel)$$

The definitions above are used for updating the current set of norms following a change of beliefs.[5] The system starts with belief base Bel, a set of abstract norms and no specific norms. The first set of active norms is then computed by setting $\Omega^S(Bel) = \big(\varnothing \cup activate(Bel)\big) \backslash \varnothing = activate(Bel)$. From then on, a new set of specific norms is computed for every new belief base generated through the belief update mechanism. In light of this, we extend Definition 5 (belief base update through action execution) as follows so that the norms in the base are also updated *after* the beliefs are updated.

Definition 19 (Normative belief base update through action execution). *Let Bel be the belief base obtained as the result of an action execution, according to Definition 5. The specific norms in Bel are updated as follows:*

$$\Omega^S(Bel) = \big(\Omega^S(Bel) \cup activate(Bel)\big) \backslash deactivate(Bel).$$

We emphasise that we use $\Omega^S(Bel)$ in general to refer to the set of active norms in Bel *after* the norm update procedure above executes, and thus assuming that the updates in Definitions 5 and 19 are executed in sequence. The only exception is the reference to $\Omega^S(Bel)$ in the right hand side of the equality above, which refers to the set of specific norms of an intermediate belief base of which only the beliefs have been updated. This is not troublesome since the belief update procedure of Definition 5 does not change the specific norms of the base. It then follows that if Bel activates ω_1^S (cf. Definition 16), then $\omega_1^S \in \Omega^S(Bel)$ and if Bel de-activates ω_2^S (cf. Definition 17), then $\omega_2^S \notin \Omega^S(Bel)$.

Notice that a specific norm remains active until a belief base update triggers the derivation of the norm's expiration condition. The update of the set of specific norms of a belief base can be computed using Algorithm 4 (defined in Section 5).

Having described how abstract norms give rise to specific norms and how they may be de-activated following a change of beliefs, we now informally explain the meaning of a specific norm ω^S within a set of beliefs Bel. If φ is an action identifier:

- $\langle \mathsf{O}_A\varphi \circ \Gamma, Exp, id\rangle \in Bel$ means that agent A is obliged to execute action φ subject to constraints Γ before a belief change operation triggers the derivation of Exp; and
- $\langle \mathsf{F}_A\varphi \circ \Gamma, Exp, id\rangle \in Bel$ means that agent A is forbidden to execute action φ subject to constraints Γ until a belief change operation triggers the derivation of Exp.

If φ is a belief:

- $\langle \mathsf{O}_A\varphi \circ \Gamma, Exp, id\rangle \in Bel$ means that agent A is obliged to hold the belief in φ subject to constraints Γ before a belief change operation triggers the derivation of Exp; and
- $\langle \mathsf{F}_A\varphi \circ \Gamma, Exp, id\rangle \in Bel$ means that agent A is forbidden to hold the belief in φ subject to constraints Γ until a belief change operation triggers the derivation of Exp.

With these in mind, we can now clarify what it means to comply with a norm. We see norms as a way of *regulating transitions between states*. Thus, when a norm becomes active, it constrains the desired or allowed transitions from the current state *onwards* until it expires. This defines the *active period* of a norm, within which its condition must be satisfied. Since we are concerned with plans, we will restrict ourselves to the transitions resulting from the execution of actions. The diagram in Figure 2 shows this in some detail, although we have omitted constraints, which can be easily understood as further restrictions on the accepted values of the instantiations.

The diagram contains a sequence of belief bases $Bel_0,\ldots, Bel_4$ in which the transitions between the bases are caused by $\gamma_1,\ldots,\gamma_4$ as indicated and α_1 and α_2 are action identifiers and β_1 and β_2 are beliefs, respectively. In the

[5] It is easy to see from Definitions 16 and 17 that the norm update definition is idempotent.

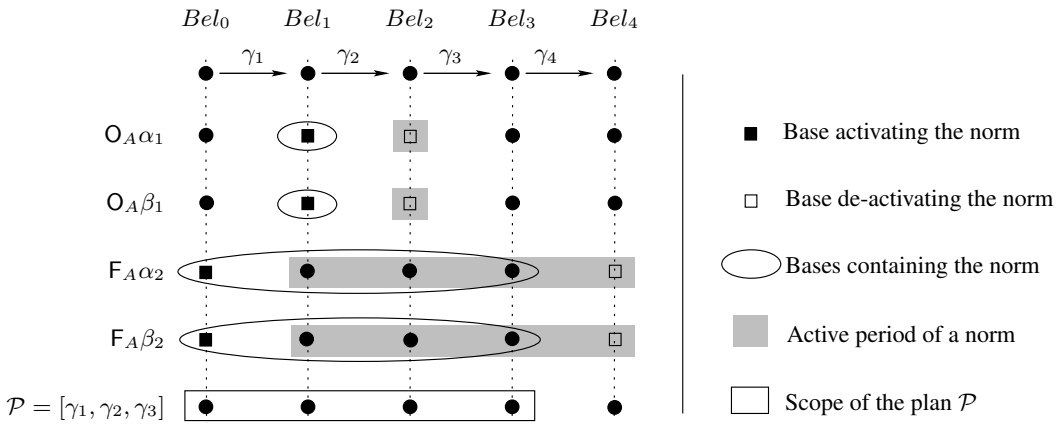

Figure 2: Norm (de-)activation periods

diagram, the grey boxes indicate the active period of the norms, *i.e.*, the belief bases that need to be checked against violations. For action norms, the check involves analysing the circumstances under which the belief bases were generated and for norms involving beliefs, which beliefs the bases support. We do not expect a norm to be fulfilled at the belief base that activates it, because this would require an agent to look ahead at the impact of an action execution not only with respect to the norms that are currently active, but also with respect to the norms that *would* be activated if the action were to be executed. In general an agent may not be in control of the circumstances that may activate a norm and hence violate it unwittingly. Conversely, once the agent becomes aware that a norm is active, it then becomes the agent's responsibility to comply with it. With this in mind, the action obligation $\mathsf{O}_A\alpha_1$ which is activated at Bel_1 specifies that from that point onwards the agent has the obligation to execute α_1. This obligation expires when the norm's expiration condition holds (in this example, the expiration condition holds in Bel_2). In practice this means that the transition from Bel_1 to Bel_2 must correspond to the execution of α_1 for the norm to be complied with, *i.e.*, that α_1 and γ_1 are unifiable. Similarly, an obligation involving the belief β_1 requires that $Bel_2 \vdash \beta_1 \cdot \sigma$, for some substitution σ. Prohibitions are treated in the same way. $\mathsf{F}_A\alpha_2$ prohibits A from executing α_2 starting from Bel_0. This prohibition expires at Bel_4, which means that none of the transitions $\gamma_1, \ldots, \gamma_4$ may correspond to α_2. The prohibition involving the belief β_2 is active from Bel_1 through to Bel_4 and hence stipulates that none of these belief bases supports it.

The impact of all this on plan evaluation can be described as follows. Suppose the plan $\mathcal{P}$ whose body contains actions $[\gamma_1, \gamma_2, \gamma_3]$ is applicable to belief base Bel_0. $\mathcal{P}$'s execution involves bases $Bel_0, \ldots, Bel_3$, so we need to check all norms that are active within this period to see if there are violations. However, the active period of each active norm may differ from $[Bel_0, Bel_1, Bel_2, Bel_3]$, *e.g.*, by starting earlier or later than Bel_0 or finishing earlier or later than Bel_3, so we need to take this into account as well. For $\mathsf{O}_A\alpha_1$ and $\mathsf{O}_A\beta_1$ above this is just $[Bel_2]$. For $\mathsf{F}_A\alpha_2$ and $\mathsf{O}_A\beta_2$ this is $[Bel_1, Bel_2, Bel_3, Bel_4]$. Even though these norms still need to be satisfied at Bel_4, the belief change to Bel_4 is not caused by $\mathcal{P}$ and hence they must not be considered during $\mathcal{P}$'s evaluation. Now, $\mathsf{F}_A\alpha_2$ and $\mathsf{F}_A\beta_2$ are activated at Bel_0, so they must belong to $\Omega^S(Bel_0)$. Thus, checking the impact of the execution of $\mathcal{P}$ on Bel_0 with respect to a set of norms requires that the specific norms in $\Omega^S(Bel_0)$ are satisfied at Bel_1; the norms in $\Omega^S(Bel_1)$ are satisfied at Bel_2 and the norms in $\Omega^S(Bel_2)$ are satisfied at Bel_3. Therefore, checking for the satisfaction of action obligations and prohibitions caused by $\mathcal{P}$ can be done by checking that the actions that resulted in $[Bel_1, Bel_2, Bel_3]$, *i.e.*, $[\gamma_1, \gamma_2, \gamma_3]$, conform with the corresponding norms. In general, all we need is the base Bel where a plan is to be executed; the sequence of actions in the plan (Definition 9) and the resulting sequence of bases generated by the execution of the plan on Bel (Definition 10).[6]

Violation of a prohibition can be detected as soon as it happens, whereas its fulfilment can only be detected once the norm expires. The treatment of obligations depends on how we interpret them. There are two options:

1. An obligation is fulfilled as soon as its specified condition is met (achievement obligations)

[6]We can also indirectly infer the sequence of actions by carefully analysing the belief changes between the bases. This is discussed further later.

2. An obligation must be fulfilled until it expires (maintenance obligations)

In this work, we only consider *achievement* obligations. *Maintenance* obligations [50] can be modelled in a straightforward way by adapting the definitions of compliance accordingly. We leave this for future work.

When a prohibition and an obligation refer to the same action or belief under overlapping periods between activation and expiration, they may come into conflict if their potential instantiations also overlap. Detecting these conflicts is non-trivial as a result of the constraint-based representation we use in this paper. When norms associated with agents simultaneously prohibit and oblige the same behaviour or effect, then a *normative conflict* arises. Such conflicts have been studied in various contexts, such as, for instance, deontic logics [61], and legal reasoning [57], as well as within the engineering of open and heterogeneous multiagent systems [60]. These situations are undesirable because whichever way software agents behave, they will violate a norm. Such conflicting norms make it impossible for agents to behave in a norm-compliant fashion. A similarly undesirable effect arises when norms associated with agents prohibit and permit the same action. While the subject of this paper is not normative conflict resolution, to make the paper self-contained we provide a brief formal account of conflict detection in AppendixB.2.

It should be noted that our main focus is to provide a mechanism for representing normative behaviour, but we do not deal with interference arising because of incorrect specification. Such interference could arise if the fulfilment of one obligation leads to the violation of another, etc. While simple interference is easy to detect (*e.g.*, between obligations of the form $\mathsf{O}\varphi$ and $\mathsf{O}\neg\varphi$), this is not necessarily the case in general. We leave the detection and management of norm interference for future work.

We have described how specific norms are added and removed due to changing beliefs, and provided an interpretation of a norm in terms of obligations and prohibitions. However, we have not yet expressed what such an obligation or prohibition might mean to an agent in terms of its effects on the agent's normative state. We address this issue next.

4.3. Norm compliance

We capture the meaning of a normative position in terms of *obligation fulfilments* and *prohibition violations*. From these we can infer whether an agent has not behaved in a norm compliant fashion [1]. Under our interpretation, an obligation is fulfilled if an obliged state of affairs or action has been brought about or executed before the obligation's expiration condition evaluates to true. Similarly, a prohibition is violated if the prohibited state of affairs or action is brought about (or executed) while the associated prohibition is active. For beliefs, fulfilment and violation can be detected by checking whether the associated belief holds at least once during the active period of the norm.

The situation with actions is potentially more complicated depending on whether we can detect action execution directly. Let us start with the indirect case. The execution of an action $\alpha = \langle \varphi, \varpi, \varepsilon^+, \varepsilon^- \rangle$ can be detected indirectly by checking its positive and negative effects, as in *see to it that* (STIT) scenarios [30]. In other words, we can assume that the action α is executed on a base Bel_i if all positive effects in ε^+ hold in Bel_{i+1} and none of the negative effects in ε^- hold.[7] To be consistent with the interpretation given to beliefs, we need to require an action under an active obligation to be executed before the obligation expires. This means that if the obligation expires at Bel_i, this base is the last admissible base for detection of the action's effects (because the action was actually executed at Bel_{i-1}, *i.e.*, before the obligation's expiration). Similarly, if an action's execution is prohibited, we need to ensure that its effects are not detected in any state in the interval starting from the state immediately after the prohibition is activated up to the point where it is de-activated, inclusively.

In our case, since we are concerned with the norm compliance of plans, we assume that the above detection can be achieved directly, by checking for the presence of the action identifier in the body of the plan – a reasonable assumption in our scenario, since the plans under evaluation are known by the agent, simplifying matters.

We can always check whether a given plan violates a prohibition, because we just need to check whether the belief or action associated with the prohibition takes place during the plan's execution. However, we can only check the *fulfilment* of a prohibition if the associated norm is de-activated during the plan's execution. Analogously, we can always check if a plan fulfils an obligation, because we just need to check whether the belief or action associated with the obligation arises during the plan's execution. However, we can only check an obligation's violation if the

[7] Note that this interpretation allows an action to *count as* some other action with respect to norms when they both have effectively the same result.

associated norm is de-activated during the plan's execution. Since we are concerned with the evaluation of the impact of plans on norms, we say that a plan *complies* with a norm if the norm is a prohibition and the plan does not violate it, or if the norm is an obligation and the plan fulfils it, because these are the conditions we can check within the period of the plan's execution.

Definition 20 (Compliance with belief obligations). *Let $BSeq = [Bel_1, \ldots, Bel_k]$ be a sequence of belief bases and $\omega^S = \langle \mathsf{O}_A\varphi \circ \Gamma, Exp, id\rangle$ a specific belief obligation. We say that*

- *ω^S is fulfilled in $BSeq$ if there are two consecutive bases Bel_i and Bel_{i+1} in $BSeq$ such that $\omega_S \in \Omega^S(Bel_i)$ and there is a substitution σ satisfying Γ such that $Bel_{i+1} \vdash (\varphi \cdot \sigma)$.*
- *ω^S is fulfilled by an agent if there is a sequence of belief bases in the agent's execution history in which ω^S is fulfilled.* □

Definition 21 (Compliance with belief prohibitions). *Let $BSeq = [Bel_1, \ldots, Bel_k]$ be a sequence of belief bases and $\omega^S = \langle \mathsf{F}_A\varphi \circ \Gamma, Exp, id\rangle$ a specific belief prohibition. We say that*

- *ω^S is violated in $BSeq$ if there are two consecutive bases Bel_i and Bel_{i+1} in $BSeq$ such that $\omega_S \in \Omega^S(Bel_i)$ and there is a substitution σ satisfying Γ such that $Bel_{i+1} \vdash (\varphi \cdot \sigma)$.*
- *ω^S is fulfilled by an agent if there is a sequence of belief bases in the agent's execution history in which ω^S is activated and de-activated and the sequence does not violate it.* □

Definition 22 (Compliance with action obligations). *Let $BSeq = [Bel_1, \ldots, Bel_k]$ be a sequence of belief bases such that each $Bel_i, 1 < i < k$, is obtained through the execution of the action γ_i on Bel_{i-1}, and $\omega^S = \langle \mathsf{O}_A\varphi \circ \Gamma, Exp, id\rangle$ be a specific action obligation. We say that*

- *ω^S is fulfilled in $BSeq$ if there is a belief base $Bel_i \in BSeq$ $(i < k)$ such that $\omega_S \in \Omega^S(Bel_i)$ and there is a substitution σ satisfying Γ such that $unify(\varphi, \gamma_{i+1}, \sigma)$ holds.*
- *ω^S is fulfilled by an agent if there is a sequence of belief bases in the agent's execution history within which ω^S is fulfilled.* □

Definition 23 (Compliance with action prohibitions). *Let $BSeq = [Bel_1, \ldots, Bel_k]$ be a sequence of belief bases such that each Bel_i $(i > 1)$ is obtained through the execution of the action γ_i on Bel_{i-1}, and $\omega^S = \langle \mathsf{F}_A\varphi \circ \Gamma, Exp, id\rangle$ be a specific action prohibition. We say that*

- *ω^S is violated within $BSeq$ if there is a belief base $Bel_i \in BSeq$ $(i < k)$ such that $\omega_S \in \Omega^S(Bel_i)$ and there is a substitution σ satisfying Γ such that $unify(\varphi, \gamma_{i+1}, \sigma)$ holds.*
- *ω^S is fulfilled by an agent if there is a sequence of belief bases in the agent's execution history within which it is activated and de-activated and the sequence does not violate it.* □

We use the above compliance definitions in the hypothetical reasoning about the effects of plans, but they can also be used in normative reasoning in general, for any given sequence of belief bases.

Our implementation of the norm compliance semantics makes use of the following observation: once a prohibition is violated, it can no longer be fulfilled; and once an obligation is fulfilled, it can no longer be violated. This means we can partially detect the normative status of a norm as soon as these conditions are detected.

Proposition 2. *Let ω^S be an obligation norm and $BSeq$ a sequence of belief bases. If ω^S is fulfilled within $BSeq$, then ω^S is fulfilled within any sequence of belief bases containing $BSeq$.*

Proof 2. *Straightforward from Definitions 20 and 22.*

Proposition 3. *Let ω^S be a prohibition norm and $BSeq$ a sequence of belief bases. If ω^S is violated within $BSeq$, then ω^S is violated within any sequence of belief bases containing $BSeq$.*

Proof 3. *Straightforward from Definitions 21 and 23.*

This *monotonicity of status* saves some computation because we can record the fact that an obligation has been fulfilled, and from then on its associated specific norm no longer needs to be checked for compliance. Analogously, we record the fact that a prohibition has been violated and its associated specific norm no longer needs to be checked. We keep this information in the base through the predicates $fulfilled(ObligId)$ and $violated(ProhibId)$. The dual of these can only be inferred once the norm expires. Thus, if a specific obligation expires and the obligation has not been fulfilled, we add $violated(ObligId)$ to the base and if a specific prohibition expires and the prohibition has not been violated, we add $fulfilled(ProhibId)$ to the base. A further benefit of this approach is that this information can also be used in the the modelling of *contrary-to-duty* norms, although we leave such considerations for future work.

Many approaches to norm management (*e.g.*, [21, 22]) assume that fulfilled norms are immediately deactivated or removed from the set of norms. However, as pointed out in [25], a norm can be violated, but need not expire.[8]

Our framework is further simplified by the fact that all active norms are stored in the agent's belief base as specific norms. The set of specific norms needs to be updated in step with the belief base as the plans execute. This means that a rigorous analysis of the normative impact of a plan before it actually executes requires a simulation of its execution.

Definition 24 (Plan compliance). *Let $\mathcal{P}$ be a plan applicable to the belief base Bel with substitution σ and $apply(\mathcal{P}, \sigma, Bel) = \langle ASeq, BSeq\rangle$, the result of this application, and ω^S be a specific norm. We say that*

- *$\mathcal{P}$ fulfils an obligation associated with a specific norm ω^S, if ω^S is fulfilled within the sequence of belief bases $[Bel] :: BSeq$ (cf. Definition 20 and 22).*
- *$\mathcal{P}$ violates a prohibition associated with a specific norm ω^S, if ω^S is violated within the sequence of belief bases $[Bel] :: BSeq$ (cf. Definition 21 and 23).* □

In order to analyse a plan $\mathcal{P}$'s compliance with a specific norm ω^S, we just have to see if ω^S is fulfilled or violated in the sequence of belief bases generated by $\mathcal{P}$'s execution together with the belief base on which $\mathcal{P}$ was executed. In practice, due to the monotonicity of status, we only need to check for the predicates $fulfilled(\omega^S)$ and $violated(\omega^S)$ in the belief base obtained at the end of $\mathcal{P}$'s execution for the specific norms in $\Omega^S(Bel) \cup \bigcup_{Bel' \in BSeq} \Omega^S(Bel')$.

Definition 24 gives the compliance status of a norm from a plan's perspective. If a plan $\mathcal{P}$ violates a prohibition ω^S, then ω^S is violated by $\mathcal{P}$. However, if $\mathcal{P}$ does not violate ω^S, this does not mean that ω^S may not be violated later. It only means that any eventual violation will not have been caused by $\mathcal{P}$. Similarly, if $\mathcal{P}$ does not fulfil an obligation ω^S, then ω^S may still be fulfilled later during some other plan's execution, as long as it is satisfied before its expiration condition holds.

Our idea is to promote plans that fulfil obligations and do not violate prohibitions because these are the conditions that we can check within the sequence of bases spanning a plan's execution. This is illustrated in the following example.

Example 3. *The initial scenario of our agent (a) was described by the following belief base, in which we include the two abstract norms introduced in Example 2.*

$$Bel_0 = \left\{ \begin{array}{l} high\_risk(1), \neg high\_risk(2), \neg high\_risk(3), \neg high\_risk(4), \neg high\_risk(5), \\ person(john), person(mary), person(susan), \\ at(john, 1), at(mary, 1), at(susan, 2), \\ emergency\_person(lee), at(lee, 3), \\ \langle \mathsf{F}_A\, evacuate(X, Y), \neg safe(Y), safe(Y), f1\rangle, \\ \langle \mathsf{O}_A\, deploy(Z, X) \circ \{X + 1 \leq Z \leq X + 3\}, high\_risk(X) \wedge clear(X), secured(X), o1\rangle, \end{array} \right\}$$

Initially, there are no specific norms. Now suppose that event $e = +level(1, medium)$ is the first one in the event queue. The event e will trigger the activation condition of the plan $\mathcal{P}$ below:

$$\mathcal{P} = \left\langle +level(X, medium), \left(\begin{array}{c} high\_risk(X) \wedge \neg high\_risk(Y) \wedge \\ \neg high\_risk(Z) \wedge (Y < Z) \end{array} \right), \left[\begin{array}{c} isolate(X), \\ evacuate(X, Y), \\ deploy(Z, X) \end{array} \right] \right\rangle$$

[8] We consider, for example, the requirement to pay a tax by a certain date. Once this date has passed, the obligation to pay the tax does not suddenly vanish. And even though once the tax is eventually paid the norm expires, this does not mean one has not violated it.

According to Definition 7, $plan\_options(e, Bel_0)$ includes the options

$$\begin{array}{lcl} o_1 & = & \langle \mathcal{P}, \{X/1, Y/2, Z/3\} \rangle \\ o_2 & = & \langle \mathcal{P}, \{X/1, Y/2, Z/4\} \rangle \\ o_3 & = & \langle \mathcal{P}, \{X/1, Y/2, Z/5\} \rangle \\ o_4 & = & \langle \mathcal{P}, \{X/1, Y/3, Z/4\} \rangle \\ o_5 & = & \langle \mathcal{P}, \{X/1, Y/3, Z/5\} \rangle \\ o_6 & = & \langle \mathcal{P}, \{X/1, Y/4, Z/5\} \rangle \end{array}$$

The execution of these options will interact in different ways with the agent's normative restrictions. In order for the agent to act in a norm-compliant way it should analyse the impact of the options before committing to any of them.

We illustrate this by showing the effects of the plan option o_1, initially considered in Example 1. The substitution for this option is $\sigma = \{X/1, Y/2, Z/3\}$. We have seen that the sequence of actions corresponding to option o_1 are $isolate(1), evacuate(1,2), deploy(3,1)$ and that the execution of $isolate(1)$ results in the addition of the beliefs $blocked(1)$ and $\neg safe(2)$. These additions trigger the activation condition of the abstract norms in Bel_0, resulting in the belief base Bel_1 below:

$$Bel_1 = \left\{ \begin{array}{l} high\_risk(1), \neg high\_risk(2), \neg high\_risk(3), \neg high\_risk(4), \neg high\_risk(5), \\ person(john), person(mary), person(susan), \\ at(john, 1), at(mary, 1), at(susan, 2), \\ emergency\_person(lee), at(lee, 3), \\ blocked(1), \neg safe(2), \\ \langle \mathsf{F}_A\, evacuate(X, Y), \neg safe(Y), safe(Y), f1 \rangle, \\ \langle \mathsf{O}_A\, deploy(Z, X) \circ \{X + 1 \leq Z \leq X + 3\}, high\_risk(X) \wedge clear(X), secured(X), o1 \rangle, \\ \langle \mathsf{F}_A\, evacuate(X, 2), safe(2), f1.1 \rangle, \\ \langle \mathsf{O}_A\, deploy(Z, 1) \circ \{2 \leq Z \leq 4\}, secured(1), o1.1 \rangle, \end{array} \right\}$$

Notice that the addition of $blocked(1)$ and $\neg safe(2)$ triggers the activation condition of the abstract norms, so their corresponding specific norms are added to the belief base Bel_1 as well. These norms now represent active constraints stipulating preferred/acceptable transitions from Bel_1.[9]

The next action in plan option o_1, i.e., $evacuate(1,2)$, is then executed. Its post-conditions are $\varepsilon^- = \{at(P,1)\} = \{at(john,1), at(mary,1)\}$ and $\varepsilon^+ = \{at(P,2)\} = \{at(john,2), at(mary,2)\}$. The execution of this action violates the prohibition $\mathsf{F}_A\, evacuate(X,2)$. The resulting belief state Bel_2 becomes:

$$Bel_2 = \left\{ \begin{array}{l} high\_risk(1), \neg high\_risk(2), \neg high\_risk(3), \neg high\_risk(4), \\ person(john), person(mary), person(susan), \\ emergency\_person(lee), at(lee, 3), \\ blocked(1), \neg safe(2), \\ at(susan, 2), at(john, 2), at(mary, 2), clear(1), \\ \langle \mathsf{F}_A\, evacuate(X, Y), \neg safe(Y), safe(Y), f1 \rangle, \\ \langle \mathsf{O}_A\, deploy(Z, X) \circ \{X + 1 \leq Z \leq X + 3\}, high\_risk(X) \wedge clear(X), secured(X), o1 \rangle, \\ \langle \mathsf{F}_A\, evacuate(X, 2), safe(2), f1.1 \rangle, \\ \langle \mathsf{O}_A\, deploy(Z, 1) \circ \{2 \leq Z \leq 4\}, secured(1), o1.1 \rangle, \\ violated(f1.1) \end{array} \right\}$$

Finally, action $deploy(3,1)$ is executed. Its scope condition instantiated with σ is $emergency\_person(P) \wedge at(P,3)$, which is satisfied for $P = lee$. Its instantiated post-conditions are therefore $\varepsilon^- = \{at(lee,3), \neg safe(2)\}$ and $\varepsilon^+ = \{at(lee,1), safe(2), secured(1)\}$. Since $2 \leq 3 \leq 4$, the obligation $o1.1$ is fulfilled with substitution $\{Z/3\}$.

[9] In this work we use the norms to *rank* plans according to their normative behaviour, but it is also possible to *block* the plans that violate active norms.

We get the following belief base Bel_3:

$$Bel_3 = \left\{ \begin{array}{l} high\_risk(1), \neg high\_risk(2), \neg high\_risk(3), \neg high\_risk(4), \\ person(john), person(mary), person(susan), \\ blocked(1), safe(2), \\ at(susan, 2), at(john, 2), at(mary, 2), clear(1), \\ emergency\_person(lee), at(lee, 1), secured(1), \\ \langle \mathsf{F}_A\, evacuate(X, Y), \neg safe(Y), safe(Y), f1 \rangle, \\ \langle \mathsf{O}_A\, deploy(Z, X) \circ \{X + 1 \leq Z \leq X + 3\}, high\_risk(X) \wedge clear(X), secured(X), o1 \rangle, \\ violated(f1.1), fulfilled(o1.1) \end{array} \right\}$$

Bel_3 satisfies the de-activation conditions of both $f1.1$ and $o1.1$, which are de-activated. Predicate $fulfilled(o1.1)$ is also added to register $o1.1$'s fulfilment.

In summary, the execution of plan option o_1 violates the prohibition $f1.1$, but fulfils the obligation $o1.1$. In Table 2 we show how each option compares with respect to the compliance of the norms in this example. ✓ indicates fulfilment and × indicates violation.[10] *The utility of each options is calculated as the total number of norm fulfilments minus the total number of norm violations incurred by each option. This will be further discussed in Section 5. As mentioned, the*

option	$\mathsf{F}_A\, evacuate(X, 2)$	$\mathsf{O}_A\, deploy(Z, 1) \circ \Gamma$	utility
$o_1 = \langle \mathcal{P}, \{X/1, Y/2, Z/3\} \rangle$	×	✓	0
$o_2 = \langle \mathcal{P}, \{X/1, Y/2, Z/4\} \rangle$	×	✓	0
$o_3 = \langle \mathcal{P}, \{X/1, Y/2, Z/5\} \rangle$	×	×	-2
$o_4 = \langle \mathcal{P}, \{X/1, Y/3, Z/4\} \rangle$	✓	✓	2
$o_5 = \langle \mathcal{P}, \{X/1, Y/3, Z/5\} \rangle$	✓	×	0
$o_6 = \langle \mathcal{P}, \{X/1, Y/4, Z/5\} \rangle$	✓	×	0

Table 2: Plan options, norm compliance and utility

execution of any plan option including action $evacuate(1, 2)$ would violate the prohibition $f1.1$, and hence the first three options would violate it. In addition, option o_2, which deploys emergency personnel from area 5 does not fulfil the obligation $o1.1$, because $2 \leq 5 \leq 4$ is not satisfied. The remaining three options do not violate the prohibition, because they evacuate people to areas 3 and 4. However, options o_5 and o_6 also deploy personnel from area 5 and hence they do not fulfil $o1.1$ either. The conclusion is that only option o_4 complies with all norms because it is the only one that fulfils the agent's obligation without violating its prohibition. ■

In the next section, we show how to use the expected normative impact of executing plan options in an augmented BDI reasoning process.

5. Augmenting Classical BDI Reasoning with Norms

We now outline the process by which norms are used in the selection of candidate plans. The idea is to augment the traditional BDI reasoning cycle with some extra steps that take the norms into account. These are depicted in the dark boxes in Figure 3. The algorithms used to realise the process are given in full detail in AppendixB.

The first set of modifications is done immediately after the belief update process. The agent's norms are updated as described in the previous section. At that point, we continue to follow the traditional BDI reasoning process. We look at the event e at the front of the event queue Ev and then try to find a plan option whose triggering condition unifies with the event and whose context condition is satisfied by the agent's belief base. We differ again by associating a *utility value* for each candidate plan satisfying the conditions. The utility is calculated as the sum of the plan's *base utility*, i.e., the utility associated with the successful achievement of goals (as instantiated by the plans used to achieve

[10] Notice that in general, it is possible for a plan option neither to fulfil nor violate a norm.

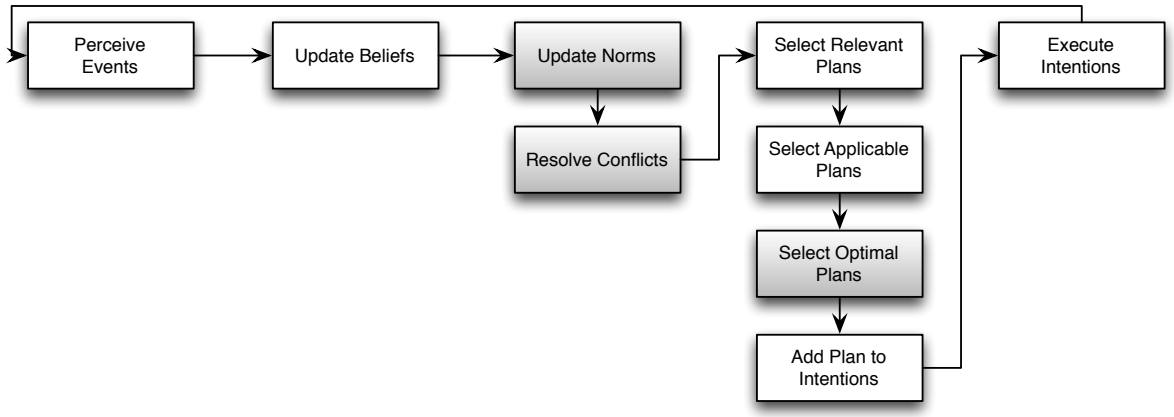

Figure 3: Control flow for the reasoning process.

them) and the plan's *benefit utility*, i.e., the utility associated with the plan's fulfilment of norms, minus the plan's *cost utility*, i.e, the utility associated with the plan's violation of norms:

$$utility(p) = baseUtility(p) + fulfilUtility(p) - violCost(p) \tag{3}$$

For simplicity, we define

$$fulfilUtility(p) = f \times fw$$

and

$$violCost(p) = v \times vw$$

where f is the total number of norm fulfilments incurred by the plan option p; fw is the *fulfilment weight* of a norm; v is the total number of norm violations incurred by p; and vw is the *violation weight* of a norm.

We then select the plan with the *highest* utility and insert its body into the intention stack and the reasoning cycle continues as in a traditional BDI framework.

Obviously, many variations of the strategy used above are possible. By fine-tuning the utility parameters above it is possible to constrain the agent's behaviour so as to force it to operate in a fully norm compliant way. It should be easy to see that if the base utility of all plans is set to 0 and the benefit utility of norm fulfilment is also set to $fw = 0$, then a plan option will only comply with all norms if its utility is non-negative. It is then possible to disregard a plan as soon as it is found to have a negative utility and only in the worst case scenario do all plan steps have to be executed to detect a violation. Although this variation allows an agent to operate without any foreseeable violations of norms, it gives the agent no choice regarding fully compliant alternatives or indeed the ability to override compliance with norms altogether, which in some cases may be the only way to achieve certain goals. A less strict approach would be the definition of a limit on the maximum number of violations an option is allowed before it is discarded.

The selection of the plan option with the highest utility can be viewed as a search through the space of all possible applicable plan options. This search can be optimised in a number of different ways, for instance by arranging the evaluation of options in a way such that the computation of identical steps in different options is simulated only once.

6. Related Work

Norms provide useful abstractions with which to study, design and implement large-scale, open and heterogeneous distributed systems. Extending BDI models with normative concerns has attracted much attention; ours is not the first proposal of a BDI agent architecture that factors in norms (*e.g.*, [6, 14, 17, 62]), to some degree. 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 work.

Our proposal bears similarities with the NoA (Normative Architecture) presented in [37], comprising a language for the specification of normative concepts and a programming language for norm-governed reasoning agents. The research is a useful and novel exercise in modelling normative aspects and proposing an extension to BDI to handle these. NoA is described informally via diagrams and English text, with Java used to provide the operational meaning of constructs – the lack of an abstract and more precise description of NoA is explicitly mentioned in [37, Chapter 3, Page 77] as a shortcoming of the research. The approach is illustrated via examples from a "blocks world", and simple plans; however, the proposal has no evaluation, no testing has been reported and there are no explicit guarantees (namely, termination, complexity and correctness) for the proposed mechanisms. Contrastingly, ν-BDI is described via algorithms (with guarantees) detailing how information/knowledge is manipulated; this increases the transparency of the proposal, thus allowing its evaluation, extension and adaptation by others. Moreover, ν-BDI allow norms of finer granularity to be specified via constraints.

The BOID framework [10] 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. Attempts at implementing BOID architectures usually make use of propositional rules, containing a set of antecedents and one conclusion. Thus for example, an obligation "if there is a fence, and it is made of wood, then it must be painted white", could be represented as

$$isFence, fenceMadeOfWood \rightarrow fenceIsWhite$$

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.

Reasoning in BOID then occurs by repeatedly applying any applicable rules to a set of input facts. Whenever a rule is applicable (due to its antecedents appearing in the set of facts), its conclusion is added to the fact base, and the process is rerun. Eventually, no new facts can be derived, and the resulting fact base should contain propositions reflecting agent action, which can then be passed onto a planner for scheduling. While conceptually elegant, BOID has not been widely used in practical systems, mainly due to a lack of robust implementations (contrasted by AgentSpeak(L) for BDI type architectures); computational complexity issues when dealing with descriptive languages; and finally, difficulties for the designer in understanding how complex sets of rules may interact.

On a more practical side, Governatori and Rotolo [26] define the BIO-logical agent framework with similar properties the BOID architecture. This framework reduces the BDI model into *Beliefs*, *Intentions* and *Obligations* and uses *propositional defeasible logic* to model the relation between the agent's mental states, and how an agent's intentions relate to normative stipulations. An important result of the BIO-logical framework is proof that formulae for intended, believed and obliged properties can be derived in linear time. As such, whereas this framework allows reasoning over the exact implications of norms on an agent's intention and vice-versa, our work is more focused on the pragmatics of filtering plan adoption in response to normative stipulations. Nevertheless, this framework is limited to a propositional representation, and thus has limited applicability to practical domains, where the power of the AgentSpeak(L) language is required.

Normative AgentSpeak(L) is an interpreter created by Meneguzzi and Luck [43] that processes norms and modifies an agent's plan library to enforce compliance with norms. This interpreter is an extended version of an interpreter of the traditional AgentSpeak(L) agent programming language that includes algorithms and meta-level actions that allow an agent to scan its own plan library for plans that would violate a set of previously accepted norms, modifying it so that the agent complies with them.

In Normative AgentSpeak(L), norms are received by an agent as regular perceptions, so they can change throughout an agent's lifecycle. Once norms are perceived, agents can choose to ignore norms and do nothing about them, or to modify their behaviour in response to norms coming into force, in which case norm processing ensues. Here, norm processing comprises the suppression of plans that violate prohibitions as well as the introduction of new plans aiming to accomplish obligations when they come into force. Modifications to the plan library are conditional to the activation and expiration conditions of each norm, in such a way that the algorithms rely on an operational semantics of norms. The algorithms for norm processing used in Normative AgentSpeak(L) are geared towards practical implementation and, in fact, an open source implementation of this system is available. However, the type of norms considered for AgentSpeak(L) are very coarse, so that norms can only refer to either very specific instances of plans or to all instances of a certain plan, which proves to be a serious obstacle to wide adoption of this particular implementation of normative agent.

In our approach, although norms are stored together with other beliefs, they can be retrieved in a straightforward manner via the $getNorms^A$ and $getNorms^S$ functions (obtaining, respectively, abstract and specific norms from the set of beliefs). The rationale for keeping norms in the belief base is to avoid creating another information/knowledge repository, thus keeping our proposal as close to the original BDI model as possible, and allowing norms to be manipulated by standard AgentSpeak machinery, with plans possibly adding/removing norms. Algorithm 4 could be extended so as to allow arbitrary normative reasoning to be carried out. For instance, conflict detection among active specific norms and the resolution of such conflicts using, for instance, the mechanism detailed in [60].

Recent efforts on the implementation of agent organisations have resulted in the $\mathcal{M}$OISE$^+$ [32] model, which includes of an organisational modelling language that includes constructs for structural, functional and deontic elements [31]. Structurally, a $\mathcal{M}$OISE$^+$ organisation comprises roles, groups and links, so that an agent associated to a *role* within a *group* is expected to accomplish certain tasks towards the achievement of collective goals. The functional element in a $\mathcal{M}$OISE$^+$ organisation describes how collective goals are accomplished through multi-agent plans that assign *missions* to individual agents. Finally, the deontic element specifies the *obligations* and *permissions* that an agent takes on when assigned roles and missions. Here, groups can be seen as *macro*-agents with their own goals and plans that are linked to individual agents through roles and missions, while using norms (the deontic specification) to guide agents towards the achievement of the collective goals. Thus, $\mathcal{M}$OISE$^+$ 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$^+$ [32], this work considers a domain-specific individual agent as largely transforming organisational events into agent plan invocations in Jason [7], 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 a certain behaviours, whereas our work defines generic algorithms for dealing with both obligations and prohibitions in a language-agnostic manner.

The work of Governatori et al. on business processes bears some resemblance to the work carried out here; [27] provides a propositional model of norms and utilises an operator to represent a contrary-to-duty (CTD) relationship between these norms. Conditionals bringing norms into force, removing norms, etc are not well catered for; instead, [27] focuses on dealing with reparation chains (i.e. those norms that come into force in sub-ideal conditions). In this paper, we concentrate on situations in which norms come into force or cease to have an effect on the agent. It should be noted that unlike [27], we are able to provide *partial reparation* for a violation by assigning a small positive utility to a contrary-to-duty obligation which comes into force due to a norm's violation. That is, we can represent partial compliance with a norm as the fraction of utility that an agent can garner by executing a plan in compliance with any CTDs, as opposed to the utility gained by complying with the initial norm.

Criado et al. [13, 14, 15] propose MaNEA (Magentix2 Norm-Enforcing Architecture), a distributed architecture based on the Magentix2 platform [58], enabling agents to handle uncertainty (of facts about the world), norm salience (*i.e.*, its importance for a given context) and norm relevance (*i.e.*, whether a norm is applicable or not to a given context) 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, as these are numeric values whose aggregated/combined effect may be unpredictable. Additionally, the approach is aimed only at agents developed in Magentix2.

Alechina et al. [2] describe N-2APL, a language based on 2APL aimed at providing an agent with norm awareness. This language does not concern itself with conditional obligations or prohibitions, and requires prohibitions to have no deadline. They then allow an agent to reason about which goals to achieve, and norms to meet, based on some priority value for the norm. Due to tractability concerns, they aim to ensure that an agent always executes plans that maximise the preferences over these priorities, by identifying a distinct set of atomic and non-atomic plans that must be executed. No decomposition from non-atomic to atomic plans is given, and the assumption is made that no two atomic plans are interleaved. There are no CTDs. N-2APL is tractable, in the sense that given a set of plans, deciding what plans to execute is not in NP. Unlike ν-BDI, N-2APL identifies all those plans that should be executed based on deadlines and priorities, rather than on world state. It thus serves as a starting point for filtering possible plans, utilising standard 2APL semantics (vvi a PG-rules) to identify what plans to execute. On the other hand, ν-BDI utilises forward simulation to identify the effects of norms on the agent, explicitly catering for CTDs etc, ensuring tractability by potentially placing a limit on search depth.

Panagiotidi *et al.* [52] proposes a normative planning framework that formalises normative reasoning in terms of planning problems specified in PDDL 2.1 [20] with extensions that allow constraints to be imposed on "desirable"

plans. The norm representation used in this framework is similar to much existing work on normative reasoning, including conditions for activation and expiration to determine when a norm is in force, and a maintenance condition that, together with a norm modality divides the state space into norm-compliant and norm-violating. Interestingly, their language allows references to violations in the activation condition of a norm, thus allowing the representation of "repair norms" (contrary to duty norms). Reasoning within the framework relies on a planner (namely, Metric-FF [29]) capable of optimising over a linear cost function specified in terms of constraints on the states achieved along each plan. These constraints refer to both achieving a goal and minimising the number of norms left in a violated and unrepaired states. This framework was later refined [51] to allow the definition of *target agents* and *timeouts*, simplifying the deontic semantics to assume that all maintenance conditions as implicitly obligations (and thus ignoring permission-type norms). Compared to ν-BDI, the framework of Panagiotidi et al. [52, 51] allows CTDs and reasoning about norms referring to world states. However, the underlying classical planning framework is known to be NP-complete for some types of domains [24, Chapter 3] (when the domain operators allow negative precondition), which ν-BDI avoids by performing forward simulation with a fixed horizon (the size of the BDI plans).

We summarise the main aspects of the related work in Table 3, below. Specifically, we compare three dimensions of each framework discussed in this section, namely: the base system upon which norm reasoning is built (the base agent system where appropriate); the language in which norms are specified; and the type of norm reasoning performed by the system. Note that when systems use a subset of Prolog as its main language, we consider its expressivity as Horn-Clause Logic, whereas when systems use Ground Horn-Clause Logic, this refers to the fragment of first-order logic accepted by planning algorithms. Finally, we highlight the main references that detail these frameworks.

Framework	Base system	Normative language	Norm reasoning	References
NoA	NoA	Horn-Clause Logic	JAVA-described	[37]
ν-BDI	AgentSpeak	Horn-Clause Logic	State-based	This paper
BOID	Propositional logic	Propositional Logic	State-based	[10]
BIO	Prop. defeasible logic	Prop. defeasible logic	Proof-theoretical	[26]
Normative AgentSpeak	AgentSpeak	Horn-Clause Logic	Event-based	[43]
MaNEA	Magentix2	First-Order Logic	Rule-Based	[13, 14, 15]
N-2APL	2APL	Propositional Logic	Deadlines and priorities	[2]
Panagiotidi *et. al*	2APL/PDDL 2.1	Ground Horn-Clauses	State-based	[52]

Table 3: Summary of related frameworks compared to ν-BDI.

7. Conclusions, Discussions and Future Work

While there has been much existing work on norms, some of it in the context of the BDI reasoning framework, this has been limited in several ways. In particular, it typically requires changes to the planning process itself, modifying the underlying reasoning framework and, one might argue, changing the nature of the BDI architecture. In response, in this paper we have proposed an extension to the BDI reasoning framework that allows the representation of normative constraints to guide the existing plan selection procedure of BDI interpreters. Our extension provides a mechanism to evaluate the normative effect of plan execution and a procedure to rank plans according to their norm compliance.

The extension, which we call ν-BDI, tackles various technical challenges implicit in the treatment of norms within an agent architecture and offers a number of benefits over alternative approaches to normative reasoning, namely: *i)* norms can be represented schematically, meaning that the periods in which a given norm is in force are determined dynamically depending on the satisfaction of the activation and de-activation conditions of the norm; *ii)* the scope of a norm's applicability can also be determined dynamically through the use of constraints; *iii)* norm satisfaction does not restrict the planning process but rather guides it, by giving priority to plans according to their norm compliance. This is important in cases where goal achievement without norm violation would not be possible; and *iv)* no modifications to the existing components of a BDI interpreter are required other than the ones regarding the ranking and selection of plans.

More specifically, the contributions here include the provision of algorithms to extend a traditional BDI interpreter, such as dMARS [18], in order to realise ν-BDI. Importantly, these algorithms can be easily adapted to any generic

BDI interpreter. We have tested ν-BDI with a hypothetical emergency evacuation scenario and claim that through algorithmic analysis and empirical testing, the norm management mechanism proposed in this paper and the algorithms used to implement it keep the computational cost of normative reasoning within reasonable bounds.

As stated previously, enabling an agent to undertake goal directed action while considering whether to comply with, or violate, a set of norms increases the robustness of the system as a whole. Norms ensure that the system behaves as specified under normal circumstances, while agents are still able to achieve their goals in exceptional situations. Such flexibility is important in many real world domains. As a concrete example, consider a set of autonomous vehicles working together to identify, triage and evacuate victims from the scene of a natural disaster. Different types of vehicles have different goals depending on their capabilities. However, they are all governed by norms to ensure that they can cooperate: UAVs have norms describing what altitudes and directions they *may*, *should* or *should not* fly at; naval vehicles should adhere to the laws of the sea (which, for example, specify which way they should turn in case of a collision), and so on. These norms ensure that all agents within the system function as a coherent whole. However, if, for example, a critically ill survivor is discovered, some of the vehicles may risk violating the norms in order to perform a rescue. The latter goal outweighs the system's norms, and the resulting goal-directed — but norm-violating — behaviour is desirable.

The use of norms within a BDI system also enables another type of flexibility in that agents can be moved from one system to another with no change in their underlying programming (i.e., no additional plans need be specified). Such agents will pursue the same goals in different systems, but are affected by the change in context described by norms [49]. A concrete example of this involves agents taking part in different electronic marketplaces. Such marketplaces would have different requirements for parameters such as how quickly somebody should be paid, the amount of leverage an agent can utilise, and so forth. The programming effort involved in moving agents between different marketplaces, the rules of which are described using norms, is greatly reduced when compared to other approaches.

This work paves the way for a number of interesting future investigations. Our mechanism substantially enhances the way in which norm-driven agents are specified, however, the norm representation used here is limited to atomic deontic formulae. Instead, the use of arbitrary deontic formulae would allow expression of norms such as "agent a is obliged to perform φ_1 and φ_2". Although we leave this for future work, the treatment of such formulae can be achieved by adding a preliminary stage whereby deontic reasoning (such as, for instance, [4, 5, 39]) would yield atomic formulae which would then be processed as proposed in this paper. However, an interesting issue arises from the interaction of logical reasoning and constraint satisfaction: the use of arbitrary deontic formulae with constraints requires deontic reasoning interleaved with constraint satisfaction. Our initial view is that a deontic formula of the type $\mathsf{X}\Phi\circ\Gamma$, where Φ is an arbitrary first-order formula, X is a deontic modality and Γ are constraints, can be dealt with by distributing constraints over Φ's subformulae as in [11]. The interaction of the deontic modalities with the logical connectives in Φ would thus need to reflect an appropriate deontic approach [30]. Finally, we envision a number of further extensions for the norm processing capabilities of the interpreter described in this paper. For instance, we have not dealt directly with conflicts arising from incorrect norm specification, although this would be desirable. Moreover, since BDI agents assume the world is non-deterministic, we also envision extending the utility-based plan selection algorithms to take a probabilistic model of the environment to further improve the chances of success and compliance of plan options.

Acknowledgements

F. Meneguzzi thanks Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS, Brazil) for the financial support through the ACI program (Grant ref. 3541-2551/12-0) and the ARD program (Grant ref. 12/0808-5), as well as Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) through the Universal Call (Grant ref. 482156/2013-9) and PQ fellowship (Grant ref. 306864/2013-4). N. Oren and W. W. Vasconcelos acknowledge the support of the Engineering and Physical Sciences Research Council (EPSRC, UK) within the research project "Scrutable Autonomous Systems" (SAsSY[11], Grant ref. EP/J012084/1).

[11] `http://www.scrutable-systems.org`

[1] Aldewereld, H., Dignum, F., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A., Sierra, C.: Operationalisation of norms for usage in electronic institutions. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 223–225. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1160633.1160669

[2] Alechina, N., Dastani, M., Logan, B.: Programming norm-aware agents. In: W. van der Hoek, L. Padgham, V. Conitzer, M. Winikoff (eds.) AAMAS, pp. 1057–1064. IFAAMAS (2012)

[3] Apt, K.R.: From Logic Programming to Prolog. Prentice-Hall, U.K. (1997)

[4] Artosi, A., Cattabriga, P., Governatori, G.: KED: A deontic theorem prover. In: Workshop on Legal Application of Logic Programming, pp. 60–76. IDG (1994)

[5] Balbiani, P.: Logical approaches to deontic reasoning: From basic questions to dynamic solutions. Int. J. Intell. Syst. **23**(10), 1021–1045 (2008). DOI http://dx.doi.org/10.1002/int.v23:10

[6] Beavers, G., Hexmoor, H.: Obligations in a BDI agent architecture. In: Proceedings of The International Conference on Artificial Intelligence, pp. 1334–1340. Las Vegas, Nevada, USA (2002)

[7] Bordini, R.H., Hübner, J.F.: BDI Agent Programming in AgentSpeak Using Jason. In: Proceedings of Computational Logic in Multi-Agent Systems, 6th International Workshop, *LNCS*, vol. 3900, pp. 143–164. Springer-Verlag (2006)

[8] Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. Wiley (2007)

[9] Bratman, M.E.: Intention, Plans and Practical Reason. Harvard University Press, Cambridge, MA (1987)

[10] Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., van der Torre, L.: The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In: Proceedings of the Fifth International Conference on Autonomous Agents, pp. 9–16 (2001). DOI http://doi.acm.org/10.1145/375735.375766

[11] Bürckert, H.: A resolution principle for constrained logics. Artificial Intelligence (66), 235–271 (1994)

[12] Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: Jack intelligent agents - components for intelligent agents in java. AgentLink Newsletter (1999). White paper, http://www.agent-software.com.au.

[13] Criado, N.: Using norms to control open multi-agent systems. Ph.D. thesis, Universitat Politécnica de València (2012)

[14] Criado, N., Argente, E., Botti, V.: A BDI architecture for normative decision making (extended abstract). In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, pp. 1383–1384. International Foundation for Autonomous Agents and Multiagent Systems (2010)

[15] Criado, N., Argente, E., Noriega, P., Botti, V.: MaNEA: A distributed architecture for enforcing norms in open MAS. Engineering Applications of Artificial Intelligence **26**(1), 76 – 95 (2013). DOI 10.1016/j.engappai.2012.08.007

[16] Dastani, M., van Riemsdijk, B., Dignum, F., Meyer, J.J.C.: A programming language for cognitive agents goal directed 3APL. In: Proceedings of the International Workshop on Programming Multiagent Systems Languages and Tools, *LNCS*, vol. 3067, pp. 111–130. Springer-Verlag (2004)

[17] Dignum, F., Morley, D., Sonenberg, E., Cavedon, L.: Towards socially sophisticated BDI agents. In: Proceedings of the ICMAS 2000, pp. 111–118. IEEE Computer Society (2000)

[18] d'Inverno, M., Luck, M., Georgeff, M., Kinny, D., Wooldridge, M.: The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System. Autonomous Agents and Multi-Agent Systems **9**(1 - 2), 5–53 (2004)

[19] Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer-Verlag, New York, U.S.A. (1990)

[20] Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research **20**, 61–124 (2003)

[21] García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.W.: Constraint Rule-Based Programming of Norms for Electronic Institutions. Journal of Autonomous Agents & Multiagent Systems **18**(1), 186–217 (2009)

[22] García-Camino, A., Rodríguez-Aguilar, J.A., Vasconcelos, W.: A distributed architecture for norm management in multi-agent systems. In: Proceedings of the Workshop on Coordination, Organization, Institutions and Norms in Agent Systems (2007)

[23] Gärdenfors, P.: Belief Revision, *Cambridge Tracts in Theoretical Computer Science*, vol. 29. Cambridge University Press (2003)

[24] Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Elsevier (2004)

[25] Governatori, G., Hulstijn, J., Riveret, R., Rotolo, A.: Characterising deadlines in temporal modal defeasible logic. In: M.A. Orgun, J. Thornton (eds.) Australian Conference on Artificial Intelligence, *LNCS*, vol. 4830, pp. 486–496. Springer (2007)

[26] Governatori, G., Rotolo, A.: BIO logical agents: Norms, beliefs, intentions in defeasible logic. Autonomous Agents and Multi-Agent Systems **17**(1), 36–69 (2008)

[27] Governatori, G., Rotolo, A.: How do agents comply with norms? In: Web Intelligence/IAT Workshops, pp. 488–491. IEEE (2009)

[28] Grossi, D., Aldewereld, H., Dignum, F.: Ubi lex, ibi poena: Designing norm enforcement in e-institutions. In: Proceedings of the Workshop on Coordination, Organization, Institutions and Norms in agent systems, *LNCS*, vol. 4386, pp. 101–114. Springer (2007)

[29] Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14**, 253–302 (2001)

[30] Horty, J.F.: Agency and Deontic Logic. Oxford University Press, Oxford (2001)

[31] Hübner, J., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents. Autonomous Agents and Multi-Agent Systems **20**(3), 369–400 (2010). DOI http://dx.doi.org/10.1007/s10458-009-9084-y

[32] Hubner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. International Journal of Agent-Oriented Software Engineering **1**(3/4), 370–395 (2007). DOI http://dx.doi.org/10.1504/IJAOSE.2007.016266

[33] Ingrand, F.F., Chatila, R., Alami, R., Robert, F.: PRS: A high level supervision and control language for autonomous mobile robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 43–49. Minneapolis, USA (1996)

[34] Ingrand, F.F., Georgeff, M.P., Rao, A.S.: An architecture for real-time reasoning and system control. IEEE Expert, Knowledge-Based Diagnosis in Process Engineering **7**(6), 33–44 (1992)

[35] Jaffar, J., Maher, M.J., Marriott, K., Stuckey, P.J.: The Semantics of Constraint Logic Programs. Journal of Logic Programming **37**(1-3), 1–46 (1998)

[36] Jones, A.J.I., Sergot, M.: Deontic Logic in Computer Science: Normative System Specification, chap. Chapter 12: On the characterisation of law and computer systems: the normative systems perspective, pp. 275–307. Wiley Professional Computing Series. Wiley (1993)
[37] Kollingbaum, M.: Norm-governed practical reasoning agents. Ph.D. thesis, University of Aberdeen (2005)
[38] Kollingbaum, M.J., Norman, T.J.: Norm adoption in the NoA agent architecture. In: Proceedings of the 2nd International Joint Conference on Autonomous Agents & Multi-AgentSystems. ACM, U.S.A, Melbourne, Australia (2003)
[39] Lee, R.M., Ryu, Y.U.: DX: A deontic expert system. J. Manage. Inf. Syst. **12**(1), 145–169 (1995)
[40] Lomuscio, A., Sergot, M.: Deontic interpreted systems. Studia Logica **75**(1), 63–92 (2003). DOI http://dx.doi.org/10.1023/A:1026176900459
[41] Lopez y Lopez, F., Luck, M., d'Inverno, M.: A normative framework for agent-based systems. In: Proceedings of the First International Symposium on Normative Multi-Agent Systems (2005)
[42] Marx, M.: Complexity of modal logic. In: P. Blackburn, J.V. Benthem, F. Wolter (eds.) Handbook of Modal Logic, *Studies in Logic and Practical Reasoning*, vol. 3, pp. 139–179. Elsevier (2007). DOI 10.1016/S1570-2464(07)80006-1
[43] Meneguzzi, F., Luck, M.: Norm-based behaviour modification in BDI agents. In: Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems, pp. 177–184 (2009)
[44] Meneguzzi, F., de Silva, L.: Planning in BDI Agents: A survey of the integration of planning algorithms and agent reasoning. The Knowledge Engineering Review p. (to appear) (2013)
[45] Meneguzzi, F., Zorzo, A.F., da Costa Móra, M., Luck, M.: Incorporating planning into BDI agents. Scalable Computing: Practice and Experience **8**, 15–28 (2007)
[46] Modgil, S., Faci, N., Meneguzzi, F., Oren, N., Miles, S., Luck, M.: A framework for monitoring agent-based normative systems. In: Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems, pp. 153–160 (2009)
[47] Nilsson, U., Maluszynski., J.: Logic, Programming and Prolog. John Wiley & Sons Ltd. (1995)
[48] Oren, N., Vasconcelos, W., Meneguzzi, F., Luck, M.: Acting on norm constrained plans. In: Computational Logic in Multi-Agent Systems, 11th International Workshop, no. 6814 in LNCS, pp. 347–363 (2011)
[49] Oren, N., Vasconcelos, W.W., Meneguzzi, F., ichael Luck, M.: Acting on norm constrained plans. In: Proceedings of the 12th International Workshop on Computati onal Logic in Multi-Agent Systems, *Lecture Notes in Artificial Intelligence*, vol. 6814, pp. 347–363. Springer (2011).
[50] Pacheco, O., Carmo, J.: A Role Based Model for the Normative Specification of Organized Collective Agency and Agents Interaction. Autonomous Agents and Multi-Agent Systems **6**(2), 145–184 (2003)
[51] Panagiotidi, S., Álvarez-Napagao, S., Vázquez-Salceda, J.: Towards the norm-aware agent: Bridging the gap between deontic specifications and practical mechanisms for norm monitoring and norm-aware planning. In: T. Balke, F. Dignum, M.B. van Riemsdijk, A.K. Chopra (eds.) COIN@AAMAS/PRIMA, *Lecture Notes in Computer Science*, vol. 8386, pp. 346–363. Springer (2013)
[52] Panagiotidi, S., Vázquez-Salceda, J., Dignum, F.: Reasoning over norm compliance via planning. In: H. Aldewereld, J.S. Sichman (eds.) COIN@AAMAS, *Lecture Notes in Computer Science*, vol. 7756, pp. 35–52. Springer (2012)
[53] Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: J. Allen, R. Fikes, E. Sandewall (eds.) Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, pp. 473–484. Morgan Kaufmann Publishers, San Mateo, CA, USA (1991)
[54] Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press, Cambridge, Massachusetts (2001)
[55] Rodrigues, O., Benevides, M.R.F.: Belief revision in pseudo-definite sets. In: Proceedings of the 11th Brazilian Symposium on Artificial Intelligence (SBIA) (1994)
[56] Sardiña, S., Padgham, L.: A bdi agent programming language with failure handling, declarative goals, and planning. Autonomous Agents and Multi-Agent Systems **23**(1), 18–70 (2011). DOI 10.1007/s10458-010-9130-9.
[57] Sartor, G.: Normative Conflicts in Legal Reasoning. AI & Law **1**(2-3), 209–235 (1992)
[58] Such, J.M., García-Fornes, A., Espinosa, A., Bellver, J.: Magentix2: A privacy-enhancing agent platform. Engineering Applications of Artificial Intelligence (2012). DOI 10.1016/j.engappai.2012.06.009. Available on-line 9 July 2012
[59] Thomas, M., Vollmer, H.: Complexity of non-monotonic logics. Computing Research Repository **abs/1009.1990** (2010)
[60] Vasconcelos, W.W., Kollingbaum, M.J., Norman, T.J.: Normative conflict resolution in multi-agent systems. Autonomous Agents and Multi-Agent Systems **19**(2), 124–152 (2009). DOI http://dx.doi.org/10.1007/s10458-008-9070-9
[61] von Wright, G.H.: An Essay in Deontic Logic and the General Theory of Action. North-Holland Publishing Company (1968)
[62] Yan-bin, P., Gao, J., Ai, J.Q., Wang, C.H., Hang, G.: An extended agent BDI model with norms, policies and contracts. In: 4th International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1–4 (2008). DOI 10.1109/WiCom.2008.1197

AppendixA. Basic Definitions

To underpin our discussion and to make the paper self-contained, we introduce some notation and definitions. We use first-order constructs for various elements of the agents and norms. Our first-order logic language is constructed from a set of n-ary predicate symbols, logical connectives and terms, which are defined as follows.

Definition 25 (Term). *A* term*, denoted generically as τ, is a variable; a constant; or a n-ary function symbol applied to (possibly nested) terms $\tau_1, \ldots, \tau_n$.*[12] □

Definition 26 (Atomic and Well-formed formulae). *An atomic formula φ is any construct of the form $p_i^n(\tau_1, \ldots, \tau_n)$, where p_i^n is a n-ary predicate symbol and $\tau_1, \ldots, \tau_n$ are terms. A well-formed formula Φ is defined as $\Phi ::= \Psi \wedge \Sigma | \neg\Psi | \forall x.\Psi | \varphi$, where Ψ and Σ are well-formed formulae; φ is an atomic formula; $\wedge$ and $\neg$ denote the logical operators for conjunction and negation, respectively; and $\forall$ is the universal quantifier over some variable x.* □

We assume the usual abbreviations for the connectives $\vee$, $\exists$, $\rightarrow$, $\leftrightarrow$: $\Phi \vee \Phi'$ 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 $(\Phi \rightarrow \Phi') \wedge (\Phi' \rightarrow \Phi)$. Additionally, we may refer to a finite set of formulae $\{\Phi_1, \ldots, \Phi_n\}$ by the formula $\Phi_1 \wedge \cdots \wedge \Phi_n$, using the two notations interchangeably for clarity of presentation.

Definition 27 (Positive and Negative Literals). *A positive literal is an atomic formula and a negative literal is the negation of an atomic formula. We represent positive and negative literals generically by the symbol L, possibly subscripted. We refer to the set of variables occurring in a literal L as $vars(L)$. A* ground literal *is a literal whose terms are all constants and we represent this as $\bar{L}$.* □

We will use first-order unification [19], which is based on the concept of variable substitution, defined next.

Definition 28 (Substitution). *A substitution σ is a finite and possibly empty set of pairs x_i/τ_i, where x_i is a variable and τ_i is a term.* □

The application of a substitution σ to a term, atomic formula, or action identifer is the simultaneous replacement of the variables in the term or formula as follows.

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. $f_i^n(\tau_1, \ldots, \tau_n) \cdot \sigma = f_i^n(\tau_1 \cdot \sigma, \ldots, \tau_n \cdot \sigma)$; and
4. $p_i^n(\tau_1, \ldots, \tau_n) \cdot \sigma = p_i^n(\tau_1 \cdot \sigma, \ldots, \tau_n \cdot \sigma)$.

Substitutions can be *composed*; that is, the composition of substitutions $\sigma_1 = \{x_1/\tau_1, \ldots, x_n/\tau_n\}$ and $\sigma_2 = \{y_1/\tau'_1, \ldots, y_k/\tau'_k\}$, denoted by $\sigma_1 \cdot \sigma_2$, 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 $\{z_1, \ldots, z_m\} = \{y_1, \ldots, y_k\} \backslash \{x_1, \ldots, x_n\}$.

Definition 29 ((Most General) Unifier). *A substitution σ is a* unifier *of two terms or atomic formulae φ_1 and φ_2, denoted as $unify(\varphi_1, \varphi_2, \sigma)$, if $\varphi_1 \cdot \sigma = \varphi_2 \cdot \sigma$. A substitution σ is the* most general unifier *of φ_1 and φ_2, denoted as $\sigma = mgu(\varphi_1, \varphi_2, \sigma)$, if $unify(\varphi_1, \varphi_2, \sigma)$ and for any other unifier σ' of φ_1 and φ_2, there is a unifier σ'' such that $\sigma' = \sigma \cdot \sigma''$.*

[12] In our formal definitions, we will generally denote: variables with letters from the end of the alphabet x, y, w, z; constants with letters from the beginning of the alphabet a, b, c, d; and functions with letters from the middle of the alphabet f, g, h, all of which will be used with or without subscript.

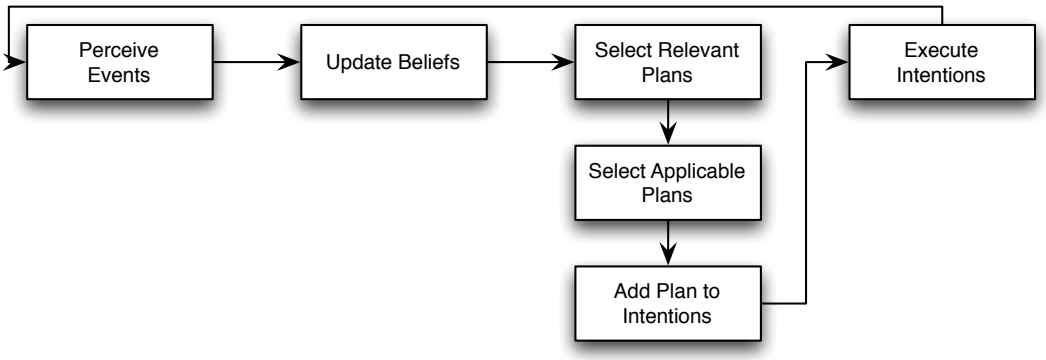

Figure B.4: Control flow for our agent interpreter.

Unification of literals is defined analogously, by ignoring the negation connective in the literals. Unification is a fundamental problem in automated theorem proving and many algorithms have been proposed [19], with recent work allowing unifiers to be efficiently computed. We assume a suitable implementation of $unify$ such that *(i)* it always terminates (possibly failing, if a unifier cannot be found); *(ii)* it is correct; and *(iii)* it has linear computational complexity.

Definition 30 (Action identifiers and beliefs). *An* action identifier *is an atomic formula and a* belief *is a ground literal.*

We assume that the sets of action identifiers and beliefs are disjoint and the availability of a sound and complete inference mechanism $\vdash$.[13] In particular, we want to know whether a formula Φ can be inferred from a set of ground formulae $\{\bar{\varphi_0}, \ldots, \bar{\varphi_n}\}$, for some substitution σ, *i.e.*, whether $\{\bar{\varphi_0}, \ldots, \bar{\varphi_n}\} \vdash \Phi \cdot \sigma$. Very importantly, in our algorithms and discussions, we always refer to the deduction mechanism "$\vdash$" and not to the semantic entailment relation "$\models$", to emphasise our focus on computational aspects.

AppendixB. Algorithms for BDI Reasoning with Norms

We now describe how the traditional BDI framework can be augmented with our normative BDI model. Our starting point is the control flow of an agent interpreter.

AppendixB.1. Agent Interpreter

A high-level description of a norm-aware BDI interpreter is shown in Algorithm 1 whose steps are presented in detail below. Each iteration of the loop (Lines 5–9) in Algorithm 1 corresponds to a reasoning cycle. Initially, new events are perceived from the environment and added to the event queue Ev (Line 5). Then, the new events are used to update the belief base Bel (Line 6). After the belief base is updated, the algorithm turns to the update of the set of norms (Line 7) and then a new plan is chosen by a new selection component that takes norm compliance into account (Line 8). Finally, an intention is selected and one of its steps executed (Line 9).

The first two mechanisms for updating events and beliefs in our interpreter are shown as Algorithms 2 and 3. Updating events (Algorithm 2) consists of gathering all the new events from the agent's sensors (Line 3), and then pushing them into the event queue (Line 4), whereas updating beliefs (Algorithm 3) consists of examining the set of events and either adding beliefs when the events are positive (that is, of the form $+\bar{\varphi}$, as captured in Line 4), or

[13] By *sound* we mean that if $\Phi \vdash \Phi'$ then $\Phi \models \Phi'$ (where "$\models$" denotes semantic entailment); by *complete* we mean that if $\Phi \models \Phi'$ then $\Phi \vdash \Phi'$. Complete and sound deduction mechanisms have a design space defined by the expressiveness of the language and complexity/decidability aspects – the more expressive the language, the fewer guarantees can be given [19]. In particular, if we assume our first-order language is restricted to Horn clauses, then we can use Prolog's resolution mechanism [3].

Algorithm 1 Norm-aware BDI agent interpreter.

1: **procedure** NUAGENTINTERPRETER($\langle Ag, Ev, Bel, Plib, Int \rangle$)
2: $IntEvs \leftarrow [\,]$ ▷ Events generated during intention execution
3: $CycEvs \leftarrow [\,]$ ▷ Events generated during a reasoning cycle
4: **loop**
5: $CycEvs \leftarrow updateEvents(IntEvs)$
6: $Bel \leftarrow updateBeliefs(CycEvs, Bel)$
7: $Bel \leftarrow updateNorms(Bel)$
8: $Int \leftarrow selectBestPlan(Ev, Bel, Plib, Int, Ag)$
9: $IntEvs \leftarrow executeIntention(Int, Bel)$

removing them when they are negative (that is, of the form $-\bar{\varphi}$, as captured in Line 5). Finally, note that we treat the event queue as a set within the belief update algorithm, and the algorithm does not modify the members of this queue. We are not concerned here with more complex belief revision mechanisms, but such an interpreter could use them [23].

Algorithm 2 Update agent events

1: **function** UPDATEEVENTS($IntEvs$)
2: $TmpEvs \leftarrow IntEvs$
3: $ExtEvs \leftarrow perceiveEvents()$
4: $pushAll(ExtEvs, TmpEvs)$
5: $pushAll(TmpEvs, Ev)$ ▷ Push all cycle events into global event stack
6: **return** $TmpEvs$

Algorithm 3 Procedure to update agent beliefs.

1: **function** UPDATEBELIEFS($CycEvs, Bel$)
2: $Bel' \leftarrow Bel$
3: **for all** events $e \in CycEvs$ **do**
4: **if** $e = +\bar{\varphi}$ **then** $Bel' \leftarrow Bel' \cup \{\bar{\varphi}\}$ ▷ add literal to belief base
5: **else if** $e = -\bar{\varphi}$ **then** $Bel' \leftarrow Bel' - \{\bar{\varphi}\}$ ▷ remove literal from belief base
6: **return** Bel'

In our approach norms are not pre-processed or translated into any intermediate format. They are simply stored in the belief base together with the agent's beliefs. However, the addition or removal of beliefs may trigger the activation or de-activation of specific norms.[14] After the agent's belief base is updated, we can then proceed to update its set of specific norms. Algorithm 4 provides the basic mechanism for this. It adds newly activated specific norms and removes expired ones. It simply implements Definitions 16, 17 and 18 and hence does not affect abstract norms or beliefs. The algorithm uses the functions $getNorms^A(Bel)$, $getNorms^S(Bel)$ and $getBeliefs(Bel)$ which, given a belief base Bel as input, returns the set of abstract norms, the set of specific norms, and the set of beliefs in Bel, respectively. The algorithm initialises working sets for these in Lines 2–5, as well as a temporary belief base Bel' initialised as the same base given as input to the algorithm. Lines 6–9 loop through all abstract norms, adding new specific norms for them as necessary. Line 10 stores the set of beliefs added during the current cycle in the variable $AddedBels$. We assume the existence of a function $getActions(CycEvs)$ that infers which actions may have been executed in the current cycle from the belief additions and deletions in $CycEvs$ (Line 11). Such a function can be easily defined by comparing the action signatures with the events in $CycEvs$. Lines 12–24 loop through all specific norms. The interaction between actions and norms regulating actions should be clear. However,

[14] For simplicity, we do not currently allow changes to the set of *abstract* norms, although this is also possible.

Algorithm 4 Update norms in belief base

1: **function** UPDATENORMS($CycEvs$, Bel)
2: $\Omega^A \leftarrow getNorms^A(Bel)$
3: $\Omega^S \leftarrow getNorms^S(Bel)$
4: $\Phi \leftarrow getBeliefs(Bel)$
5: $Bel' \leftarrow Bel$
6: **for all** $\omega^A \in \Omega^A$, ω^A of the form $\langle \nu, Act, Exp, id \rangle$, **do**
7: **for all** σ such that $\Phi \vdash Act \cdot \sigma$ and $\Phi \nvdash Exp \cdot \sigma$ **do**
8: **if** $\langle \nu, Exp \cdot \sigma, id \rangle \notin \Omega^S$ **then**
9: $Bel' \leftarrow Bel' \cup \{\langle \nu, Exp \cdot \sigma, nid \rangle\}$ ▷ add newly activated norm with a new norm identifier nid
10: $AddedBels \leftarrow \{\phi | +\phi \in CycEvs\}$
11: $\Delta \leftarrow getActions(CycEvs)$
12: **for all** $\omega^S = \langle \mathsf{X}_A \varphi \circ \Gamma, Act, Exp, id \rangle \in \Omega^S$ **do**
13: **if** $\mathsf{X} = \mathsf{F}$ **then**
14: **if** there is σ, $satisfy(\Gamma, \sigma)$, **and** $\begin{pmatrix} (\varphi \text{ is a belief } \textbf{and } AddedBels \vdash (\varphi \cdot \sigma)) \textbf{ or} \\ (\varphi \text{ is an action identifier } \textbf{and } unify(\varphi, \gamma, \sigma) \text{ for some } \gamma \in \Delta) \end{pmatrix}$ **then**
15: $Bel' \leftarrow Bel' \cup \{violated(\omega^S)\}$
16: **else if** $\mathsf{X} = \mathsf{O}$ **then**
17: **if** there is σ, $satisfy(\Gamma, \sigma)$, **and** $\begin{pmatrix} (\varphi \text{ is a belief } \textbf{and } Bel \vdash (\varphi \cdot \sigma)) \textbf{ or} \\ (\varphi \text{ is an action identifier } \textbf{and } unify(\varphi, \gamma, \sigma) \text{ for some } \gamma \in \Delta) \end{pmatrix}$ **then**
18: $Bel' \leftarrow Bel' \cup \{fulfilled(\omega^S)\}$
19: **if** there is $\sigma, \Phi \vdash Exp \cdot \sigma$, **then**
20: $Bel' \leftarrow Bel' \setminus \{\omega^S\}$ ▷ remove this expired norm
21: **if** $\mathsf{X} = \mathsf{F}$ and $violated(\omega^S) \notin Bel'$ **then**
22: $Bel' \leftarrow Bel' \cup \{fulfilled(\omega^S)\}$
23: **else if** $\mathsf{X} = \mathsf{O}$ and $fulfilled(\omega^S) \notin Bel'$ **then**
24: $Bel' \leftarrow Bel' \cup \{violated(\omega^S)\}$
25: **return** Bel'

even though a norm may not directly refer to an action identifier, an action's execution may still interact with it by adding or removing beliefs that are ruled by the norm. In general, a prohibition cannot be violated by the removal of beliefs. Similarly, obligations cannot be violated by their inclusion.[15] The addition of a belief may violate a prohibition (Line 14, top). Conversely, the fulfilment of an obligation to hold a belief does not necessarily depend on the belief's addition (as long as the belief base derives the belief – Line 17, top). With this in mind and due to the monotonicity of obligation fulfilments and prohibition violations (Propositions 2 and 3), we can record the fulfilment of an obligation and the violation of a prohibition, respectively, as soon as they are first detected and for this we use the predicates $fulfilled(Oblig)$ and $violated(Prohib)$ (Lines 13–18). The removal of expired norms is carried out in Line 20 followed by the final assessment of the normative status of the newly expired norms: if an obligation expires without fulfilment, then it has been violated, and analogously, if a prohibition expires without being violated, it has been fulfilled. Together, these ensure that the normative state of all expired norms is always known. Once all specific norms are processed, the updated belief base is returned.

The next step in the reasoning cycle is the selection of the plan with the highest utility. Based on Equation (3), Algorithm 5 works by using the number of violations and fulfilments incurred by every plan option applicable to a given event (Lines 4-6) in the calculation of the utility of the option. The algorithm then selects one of the plan options with maximum utility (Line 7). It always terminates since all of its loops are executed over finite constructs and all tests carried out in the loops terminate. Example 3 shows the utility of several plan options using base utility 0 and setting the violation and fulfilment weights to 1.

After new plans are adopted, a step from an intention is executed using $executeIntention(Int, Ev)$, detailed in

[15] In traditional BDI, since a belief base is simply a set of ground literals, derivability can be checked simply by checking membership in the

Algorithm 5 Selection of a plan with the highest utility

1: **procedure** SELECTBESTPLAN($Ev, Bel, Plib, Int$)
2: $e \leftarrow pop(Ev)$
3: $Opt \leftarrow \varnothing$
4: **for all** $p \leftarrow \langle t, c, bd\rangle \in Plib$ **do**
5: **for all** σ such that $unifies(e, t, \sigma)$ and $Bel \vdash c \cdot \sigma$ **do**
6: $Opt \leftarrow Opt \cup \langle p, \sigma\rangle$ ▷ Store the option
7: $\langle p, \sigma\rangle \leftarrow \text{argmax}_{\langle p,\sigma\rangle \in Opt}$ OPTIONUTILITY($\langle p, \sigma\rangle, Bel$) ▷ selects a plan with maximum utility
8: **if** $e = \ !g$ **then**
9: get intention $int' = \langle \sigma', st'\rangle$ that generated g
10: $bd \leftarrow$ body of p
11: $push(\langle bd \cdot \sigma\sigma'\rangle, st')$
12: $Int' \leftarrow Int$
13: **else**
14: $Int' \leftarrow Int \cup \{\langle \sigma, bd \cdot \sigma\rangle\}$
15: **return** Int'

Algorithm 6 Evaluating the normative impact of a plan option

1: **function** OPTIONUTILITY($\langle[s_1, \ldots, s_n], \sigma\rangle, Bel$)
2: $\langle v, f\rangle \leftarrow$ PLANIMPACT($[s_1, \ldots, s_n], Bel$) ▷ gets the no. of violations and fulfilments of this option
3: $u \leftarrow baseUtility([s_1 \cdot \sigma, \ldots, s_n \cdot \sigma]) + (fw \times f) - (vw \times v)$ ▷ computes the utility of this option
4: **return** u ▷ returns plan utility from violations and fulfilments

Algorithm 9. This consists of selecting one intention int_i from the intention structure, and executing the topmost step s_k of int_i's stack. If this step is an action, it is executed immediately in the environment, whereas if it is a subgoal, then an event is added to the event queue. For the purposes of this paper, an action is only considered as far as the events an agent expects to receive once an action is successfully executed. Action execution is defined in Algorithm 8.

Here, we are not concerned with action failure or success, but rather with the fact that if an agent selects a plan containing an action that affects the normative state, this must be taken into account in the normative reasoning. Thus, the effects of the action within its scope conditions are pushed onto the event queue when the action is executed (Lines 3–6).

This traditional part of this modified agent interpreter, and the processes upon which it is based, have a very low computational cost, as demonstrated by various practical implementations such as dMARS [18], PRS [33], and Jason

base. If we were to consider a more sophisticated type of negation, such as *negation by failure*, we could extend the norm reasoning procedure as proposed in [55].

Algorithm 7 Evaluating the normative impact of a plan option

1: **function** PLANIMPACT($[s_1, \ldots, s_n], Bel$)
2: $Bel' \leftarrow Bel/\{violated(\omega^S)\}$, for any norm ω^S ▷ initialise working base leaving previous violations aside
3: **for all** action steps s_i in $[s_1, \ldots, s_n]$ **do**
4: $Ev \leftarrow [\,]$
5: EXECUTEACTION(s_i, Bel', Ev) ▷ simulate step's execution on Bel'
6: $Bel' \leftarrow$ UPDATENORMS(Bel', Ev) ▷ update norms
7: $v \leftarrow |\{\omega^S \mid Bel' \vdash violated(\omega^S)\}|$
8: $f \leftarrow |\{\omega^S \mid Bel' \vdash fulfilled(\omega^S)\}|$
9: **return** $\langle v, f\rangle$ ▷ returns the total number of violations and fulfilments

Algorithm 8 Action execution.

procedure EXECUTEACTION($\langle \varphi, \varpi, \varepsilon^+, \varepsilon^- \rangle, Bel, Ev'$)
  for all σ such that $Bel \vdash \varpi \cdot \sigma$ **do**
    for all $\phi \in \varepsilon^+$ **do**
      push($+\phi \cdot \sigma, Ev'$)
    for all $\phi \in \varepsilon^-$ **do**
      push($-\phi \cdot \sigma, Ev'$)

Algorithm 9 Intention execution

function EXECUTEINTENTION(Int, Bel)
  $Ev' \leftarrow [\,]$
  $int \leftarrow peek(Int)$, where int is of the form $\langle \sigma, [s_1, \ldots, s_n] \rangle$
  if s_1 is an action $a = \langle \varphi, \varpi, \varepsilon \rangle$ **then**
    EXECUTEACTION($\langle \varphi, \varpi, \varepsilon \rangle \cdot \sigma, Bel, Ev'$)
  else if s_1 is a subgoal g **then**
    $push(+!g \cdot \sigma, Ev')$
  if $int = \langle \sigma, [s_1] \rangle$ **then**
    Remove int from Int
  else
    Replace $\langle \sigma, [s_2, \ldots, s_n] \rangle$ for int in Int
  return Ev'

[7], among others. A detailed analysis of the complexity introduced by the norm management components is left for future work.

AppendixB.2. Normative Conflict

We formally define normative conflict along the lines of the work presented in [60], extending it to cope with activation and deactivation conditions. We define conflict between two specific norms:

Definition 31 (Conflict). *Norms ω^S and $\omega^{S'}$ are in conflict under substitution σ^c, denoted as $conflict(\omega^S, \omega^{S'}, \sigma^c)$, X being O or P, if and only if:*

- $\omega^S = \langle \mathsf{F}_A \varphi \circ \Gamma, Exp, \sigma, id \rangle$, $\omega^{S'} = \langle \mathsf{X}_{A'} \varphi' \circ \Gamma', Exp', \sigma', id' \rangle$ *or*
- $\omega^S = \langle \mathsf{X}_A \varphi \circ \Gamma, Act, Exp, \sigma, id \rangle$, $\omega^{S'} = \langle \mathsf{F}_{A'} \varphi' \circ \Gamma', Act', Exp', \sigma', id' \rangle$

and the following conditions hold:

1. $unify(\langle A, \varphi \rangle \cdot \sigma, \langle A', \varphi' \rangle \cdot \sigma', \sigma^c)$
2. $satisfy(((\Gamma \cdot \sigma) \cup (\Gamma' \cdot \sigma')) \cdot \sigma^c, \sigma^s)$ *for some* σ^s

□

That is, a conflict occurs between a prohibition and an obligation if *1)* a substitution σ^c can be found that unifies the variables of the two norms (taking substitutions σ, σ' into account), and *2)* the constraints from both norms can be satisfied (taking σ^c under consideration).

Conflict detection between norms should ideally be followed by a *resolution* of the conflict, whereby norms are (semi-) automatically altered according to some principle, thus ensuring conflicts are removed. One such mechanism is defined in [60]: the proposed mechanism is fine-grained, manipulating constraints in the norms thus ensuring that any undesirable overlaps in variables' values are avoided. The mechanism is also flexible and generic enough to capture various normative conflict resolution strategies such as *lex superior* (that is, the norm established by the most

powerful authority should be preserved) and *lex posterior* (that is, the norm most recently established takes precedence over previously defined ones). In this paper, we take a utility based approach to conflict resolution. An agent will, when faced with a normative conflict, comply with the norm that results in a higher utility for it.