Modelling and monitoring interdependent expectations

Stephen Cranefield[1], Michael Winikoff[1], and Wamberto Vasconcelos[2]

[1] Department of Information Science, University of Otago, Dunedin 9054, New Zealand
[2] Department of Computing Science, University of Aberdeen, AB24 3UE, Aberdeen, UK

Abstract. Previous research on modelling and monitoring norms, contracts and commitments has studied the semantics of concepts such as obligation, permission, prohibition and commitment; languages for expressing behavioural constraints (such as norms or contracts) to be followed by agents in specific contexts; and mechanisms for run-time monitoring of fulfilment and violation of these constraints. However, there has been little work that provided all of these features while also allowing the current expectations of agents, and the fulfilment and violation of these expectations to be expressed as first-class constructs in the language. This paper demonstrates the benefits of providing this capability by considering a variety of use cases and demonstrating how these can be addressed as applications of a previously defined temporal logic of expectations and an associated monitoring technique.

1 Introduction

Much research in multi-agent systems has been influenced by organisational principles from human society, and in particular social concepts such as norms and commitments have been extensively studied due to their potential to enable the efficient specification and management of agent interaction in open societies of autonomous agents.

Previous research on modelling and monitoring norms, contracts and commitments has studied the semantics of concepts such as obligation, permission, prohibition and commitment; languages for expressing behavioural constraints (such as norms or contracts) to be followed by agents in specific contexts; and mechanisms for run-time monitoring of fulfilment and violation of these constraints. However, there has been little work that provided all of these features while also allowing the existence, fulfilment and violation of obligations and commitments to be expressed as first-class constructs in the language. We believe that the ability to directly express statements about these features of an agent's social context is important as it allows the investigation of richer types of norms and contracts that are interdependent. Our aim in this paper is to demonstrate that this is a capability that is desirable but not adequately addressed to date, and show how a logic and monitoring technique developed in our previous work can meet our requirements.

In this paper, we are not concerned with distinctions between norms and commitments, and generalise both concepts to the notion of *expectations* on future world states, events and/or agent actions, while ignoring social issues such as where these expectations come from (e.g. mandated by authorities, inferred through observation and experience, or requested and accepted via agent messaging) and how they are embedded

into the relationships that exist between agents. In our view these issues can be largely decoupled from the questions of what it means to have an expectation that is active, fulfilled or violated, and how these expectations change from one state to the next.

The structure of this paper is as follows. In Section 2 we present a survey of a range of approaches to modelling and monitoring various types of expectations. Section 3 provides an overview of our previously defined logic of expectations, and explains why previously imposed restrictions on the nesting of expectation-related modalities can be lifted. Some use cases illustrating the utility of modelling interdependent expectations are presented in Section 4, and the paper is concluded in Section 5.

2 Previous work

A wide variety of approaches have been investigated for modelling and monitoring constraints on agents' future behaviour in the context of electronic institutions, normative multi-agent systems and commitment-based semantics for agent communication. Early work in electronic institutions (e.g. [14, 18, 24]) focused on the development of middleware that can directly interpret an institution specification provided by a designer and ensure that agents follow the norms, and for this reason considered norm representations that have a procedural rather than declarative flavour, giving rise to the so-called "protocol-based norms". Work in the related field of normative multi-agent systems (e.g. [8]) has tended to focus on higher-level declarative representations of norms. Research on commitment-based semantics for agent communication (e.g. [22, 27]) aims to explain the individual speech acts and/or complete dialogs exchanged between agents in terms of the commitments requested and made by one agent towards another.

There are strong links between these research fields with much work crossing the boundaries between them, e.g. the design of a norm representation language that includes operational details such as violation checks and repair strategies alongside a declarative norm [25], the extension of e-institution middleware to handle rule-based norms as well as protocol-based norms [18], and an institution specification language that models both norms and agent communications in terms of commitments [16].

Below we discuss work in these areas, focusing on the formalisms used, whether concepts such as expectation, fulfilment and violation are expressible in those formalisms, and whether (and how) the monitoring of expectations has been addressed. The approaches discussed range from high-level logical models, investigated mainly to gain semantic understanding of norms, commitments or general expectations, to operational models that can be directly executed and are therefore amenable to run-time monitoring. However, there has been no work that provides a good semantic account of the activation, fulfilment and violation of expectations of any sort, allowing these concepts to be explicitly represented, and also providing a technique for monitoring expectations, except for the work of Governatori and Rotolo [20], which addresses only the recovery from violations via contrary-to-duty norms. In Section 3 we show how our prior work on modelling and monitoring expectations can be extended to provide all three features, and argue why this ability opens up a new range of interesting use cases in expectation modelling and monitoring.

2.1 Logical approaches

Dignum et al. [12] began a line of research investigating the extension of dynamic [12] and temporal [9] logics with deontic concepts to allow the expression of obligations involving deadlines. The obligations studied address either the performance of specific actions [12] or the fulfilment of (atemporal) propositions [9] by a deadline. The first work in this area [12] defined the semantics of formulae relative to a state and a trace so that "the history (i.e. the trace) of an ideal world might differ from the history of the present world", and this feature was used to define the notion of ideality represented by obligations. Later work used simpler semantics in which models of the logic are assumed to include a propositional constant $Viol$[3]. Broersen et al. [9] also used an ideality proposition Idl to allow a more subtle account of deadline obligations. These propositions have no semantics of their own—they are given semantics indirectly via the definitions of the obligation operators, which constrain the states in which $Viol$ and (for Broersen et al.) Idl should hold.

The works cited above did not address the modelling of obligations dependent on the fulfilment or violation of other obligations, except for the simple case of the violation or fulfilment of single actions. Also, in the examples of interdependent obligations considered in this [12] and earlier work [11], rather than explicitly using $Viol$ and Idl predicates as conditions of norms, predicates directly expressing the occurrence or lack of occurrence of the specific desired action are used. While this demonstrates how, for specific examples, an obligation can be made conditional on a predicate that happens to correspond to fufilment or violation of another obligation, there is no systematic treatment of nested violation and fulfilment operators within obligations. This is reasonable given the restricted setting (obligations to perform a given action), but this approach leaves open the question of how inter-related norms with more complex temporal structure could be expressed.

Alberti et al. [1] describe a means to perform run-time protocol compliance monitoring based on logical constraints expressing positive and negative expectations as the consequences of observed actions. At run time, agent messages are detected and asserted as facts, and abductive inference is used to keep track of pending, fulfilled and violated expectations. However, this information about the state of expectations cannot be expressed using constraints, so interdependent expectations cannot be modelled.

Verdicchio and Colombetti [26] use a variant of CTL* to provide axioms defining the lifecycle of commitments based on their making and cancelling as well as requests for them, which come about through the exchange of messages. The language includes predicates to represent a commitment being made and whether it is fulfilled, violated or pending. It seems that these predicates could appear within the content of commitments. There is no discussion of how the language could be used for practical reasoning.

Bentahar et al. [6] define model-theoretic semantics for their previously proposed Commitment and Argument Network (CAN) formalism [5] for modelling agent communication in terms of social commitments and argumentation. Their logical language can express the creation of commitments of various sorts and requests for commitments

[3] In some work it is noted that this propositional constant could be qualified, e.g. with a norm index [11], so that different types of violation can be distinguished.

to be made, as well as the satisfaction and violation of commitments. It appears that the satisfaction and violation operators can be nested within the content of a commitment. Their discussion of pragmatic aspects of their formalism [5] does not address the monitoring of commitments.

Singh [22] provides model-theoretic semantics for commitments, with two modalities defining practical and dialectical commitments between a debtor x and creditor y. The language allows these modalities to be nested. The paper discusses possible reasoning postulates and their soundness and completeness, but there is no discussion of how this logic could be used in practice. It is, however, claimed that the approach provides a basis for specifying precisely how commitments arise in a context and can be manipulated. Modalities corresponding to fulfilment and violation are not discussed.

Governatori and Rotolo [20] propose a technique for design-time checking of a set of rules (specifying some process) against a set of normative rules regulating it. This is in contrast with the run-time checking of actual behaviour that is the focus of this paper, and is therefore contingent on the process descriptions to be checked being available for this purpose. The normative language used is based on defeasible logic and has a special focus on "contrary to duty" norms. It thus has an implicit notion of violation of a norm expressible in the language.

Cranefield and Winikoff [10] define an extension of propositional linear temporal logic that includes temporal operators stating that an expectation currently exists, is fulfilled, or is violated as a result of a particular conditional rule of expectation. In contrast to the work discussed above, the concepts of violation and fulfilment of expectations are given their own first class semantics. The logic, as described previously, did not allow formulae representing existence, fulfilment or violation of an expectation to appear nested within a rule of expectation; for example, a rule could not be triggered by the violation of another rule. A model checking procedure allows the truth of these formulae to be determined either off-line (e.g., when checking an audit trail) or incrementally as new states become available. This logic is the basis of the discussion in this paper, and a modified version is described in Section 3.

2.2 Rule languages

García-Camino et al. [18] present a language for defining conditional norms and the sanctions or rewards associated when norms are fulfilled or violated. Norms control the utterance of speech acts within particular periods (specified in terms of dates or relative to other speech acts). Sanctions can modify attributes of an agent, such as its credit. The language is given an operational semantics in terms of the Jess expert system shell, and this allows norm fulfilments and violations to be detected at run-time and sanctions to be applied. However, the occurrence of fulfilments and violations cannot be expressed within the normative rules themselves.

García-Camino et al. [19] define an expressive rule language with constraints for specifying conditional norms and explicitly tracking the normative state of a multi-agent system as agents exchange messages. Rules may refer to norms, so it is possible to define rules stating, for example, that one obligation triggers another. Although the rules track the normative state of the multi-agent system and therefore detect violations of norms, these cannot be represented using the proposed set of predicates for representing

normative states. As the rule language is not dependent on the predicates used to model states, additional fulfilment and violation predicates could be added. However, the only semantics for predicates are any operational ones defined by rules.

Fornara et al. [16] describe an approach for specifying institutions in which agents communicate. The content of commitments comprise an action, proposition or referential expression existentially or universally bound to an interval of time. Norms are event-driven rules to create, update or cancel commitments. Although the authors advocate the suitability of an operational approach to checking agent norm-compliance, they do not give details as to how this could be done, and their language cannot express fulfilments and violations of commitments and norms.

Aldewereld et al. [2] have considered the use of "counts-as" predicates to link normative (abstract) events with real-world (concrete) events. In particular, obligation and prohibition norms in deontic logic are operationalised as "counts-as" statements: an obligation (respectively prohibition) with content ϕ maps to the statement that $\neg\phi$ (respectively ϕ) counts as a norm violation. Norms can have a maintenance condition (once active, the norm is deemed violated if this condition evaluates to false), and this allows a limited degree of temporal expressiveness. It is not clear if the violations of different norms can be distinguished and there is no discussion of whether the norm violation and fulfilment conditions can be used within the content of norms. The approach is implemented using the DROOLS forward-chaining engine.

2.3 Action description languages

Artikis and Sergot [3] use the event calculus for specifying and tracking normative states of multi-agents systems based on the concepts of obligation, power and permission. Their approach specifies how the actions agents perform affect the values of fluents (dynamic properties) encoding the state of the domain and the powers, permission and obligations of the actors. Obligations represent actions that agents should perform (rather than states of the world they should bring about). A violation fluent is used to declare that an action causes a violation, but this has no special semantics. Farrell et al. [15] present a similar approach for modelling and monitoring the state of contracts.

Commitment machines [27] define agent interaction protocols by specifying the preconditions and effects of the agent actions in terms of commitments that exist between participants. A set of protocol states are defined in terms of the propositions and commitments that hold in them and domain actions are defined in terms of the propositions and commitments they cause to hold (their "effects"). Actions cause transitions between states if the new state is a logical consequence of the original state and the action's effects. Agents interpret commitment machines at run time to determine a desired path through the protocol, or they can execute a finite state machine compiled from the commitment machine. There is no notion of violation of a commitment in this formalism—an execution of the protocol either reaches a state in which a desired commitment exists, or it does not. Commitments can be conditional on the existence of other commitments, but these represent instances of conditional commitments between agents that were created during the protocol execution, not general rules that one commitment should always create another.

2.4 Automata-based approaches

Spoletini and Verdicchio [23] developed an automata-based technique for monitoring commitments expressed in a propositional temporal logic with both past and future operators. The monitoring problem is modelled as a word recognition problem over an alphabet comprising propositions representing the contents of "sniffed" agent messages and the values of past-oriented subformulae of the formula to be monitored. The formula is preprocessed using Gabbay's rules [17] to separate out any future operators nested within past operators. The values of subformulae formed from past operators with no nested future operators are recognised dynamically by deterministic Büchi automata, and these subformulae are replaced by special propositions representing the outputs of the automata. The resulting formula is then translated into an alternating modulo counting automata. In this approach, fulfilment and violation are represented by the operational condition of the automaton reaching an acceptance or non-acceptance state—there is no representation of fulfilment or violation within the language used for representing commitments.

Modgil et al. [21] model norms with augmented transition networks (ATNs), comprising three states representing the norm being inactive, active and either fulfilled or violated. ATNs are processed via an architecture in which observer agents send messages to monitors, which trigger transitions in the ATNs and notify a manager agent of norm fulfilments and violations. The norms could, in principle, include messages announcing fulfilments and violations in arc labels, with the manager having the responsibility of sending these, but this extension is not proposed in the paper. The approach is defined in terms of a highly procedural account of the architecture and the interaction between its components, and it is difficult to relate it to more declarative approaches.

3 A temporal logic of expectation

The logic used in this paper is based on an extension of propositional linear temporal logic proposed by Cranefield and Winikoff [10]. However, in this paper we introduce some changes[4] from the original presentation and omit some features of the language that are not relevant to the discussion in this paper. The syntax of the logic is described by the following grammar:

$$\begin{aligned}\phi ::= {} & \mathsf{Exp}(\phi,\phi,n,\phi) \mid \mathsf{Fulf}(\phi,\phi,n,\phi) \mid \mathsf{Viol}(\phi,\phi,n,\phi) \mid \\ & \mathsf{Exp}(\phi,\phi) \mid \mathsf{Fulf}(\phi,\phi) \mid \mathsf{Viol}(\phi,\phi) \mid \\ & p \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc\phi \mid \ominus\phi \mid \phi \,\mathsf{U}\, \phi; \mid \phi \,\mathsf{S}\, \phi \mid n \mid \mathsf{Trunc_S} \mid \mathsf{Progress}(\phi,\phi)\end{aligned}$$

where p is a proposition, $\bigcirc$ is the standard temporal "next" operator, $\ominus$ is the standard temporal "previous" operator, U is the standard temporal "until", S ("since") is a

[4] The syntax of the previous version of the logic did not include the four-argument versions of Exp, Fulf, Viol nor the $\mathsf{Trunc_S}$ and Progress operators. However, these operators were defined semantically and used in the definitions of the two-argument versions of Exp, Fulf and Viol (which we have renamed here from their original names ExistsExp, ExistsFulf and ExistsViol). In this paper, to allow a concise presentation, we include all these operators in the syntax.

backwards-looking version of until, and n is a nominal: a proposition that is constrained to be true in exactly one state in the model. We assume that the model contains at least one nominal for each state, as these are used in the semantics to identify the states in which "rules of expectation" fire and introduce new expectations. Nominals are a feature of hybrid logic [7], and the original version of the logic [10] contained other hybrid logic constructs. However, only nominals are needed in this paper. The $\mathsf{Trunc}_\mathsf{S}$ and Progress operators are explained below.

We assume the propositions include $\top$ (true) and $\bot$ (false), with their usual meanings, and define as abbreviations the Boolean connectives $\vee$ and $\rightarrow$, the derived temporal operators "eventually ϕ" ($\Diamond\phi \equiv \top\,\mathsf{U}\,\phi$), and "always ϕ" ($\Box\phi \equiv \neg\Diamond\neg\phi$), and similar backwards-looking versions $\Diamondblack\phi \equiv \top\,\mathsf{S}\,\phi$ and $\boxminus\phi \equiv \neg\Diamondblack\neg\phi$.

The semantics determine the truth of a formulae at a given state in a model comprising a finite or infinite sequence of states together with a valuation function specifying the propositions that hold in each state. In the case of a finite model, either *strong* or *weak* semantics can be used to evaluate the $\bigcirc$ and U operators [13]. The strong semantics assume a formula is false if the model does not include enough states to evaluate a formula, while the weak semantics assume a formula is true in this situation. Thus, in the final state of a finite model, $\bigcirc p$ is false under the strong semantics and true under the weak semantics. The operator $\mathsf{Trunc}_\mathsf{S}$ is a simplified form of an operator defined by Eisner et al. [13], and its semantics truncate the model at the current state and use the strong semantics to evaluate its argument formula. Essentially this means to determine whether the argument formula can be known to be true without using any information in future states. Formally, $\mathsf{Trunc}_\mathsf{S}\,\phi$ is true in state i of a model $\mathcal{M}$ if and only if ϕ is strongly true ($\models^+$) in a truncated model $\mathcal{M}^i$ where all states after i have been removed:

$$\mathcal{M}, i \models \mathsf{Trunc}_\mathsf{S}\,\phi \quad \text{iff} \quad \mathcal{M}^i, i \models^+ \phi$$

3.1 Expectation Operators

The first two arguments, of the Exp, Fulf and Viol operators represent a conditional rule of expectation. Although the condition and expectation of a rule always appear as separate arguments of an operator in our logic, for convenience we will write $\lambda \Rrightarrow \rho$ as shorthand[5] for "the rule given by the pair λ and ρ". The meaning of a rule $\lambda \Rrightarrow \rho$ is that if λ evaluates to *true* in any state, given the information in the model up to that state, then ρ is an expected constraint on the model at that state.

Unlike most approaches to modelling norms and commitments, our expectations are not limited to propositions that describe a desired property of a single state (e.g. the performance of a given action by an agent) in conjunction with a simple deadline constraint. Instead we aim to study the fulfilment and violation of more general types of expectation, such as those that aren't brought about by agents' actions ("The sun will rise each morning") and those with compex temporal structure ("If I pay for a subscription then the publisher will send me a magazine issue each month for a year from the month after my payment is received"). Thus, λ and ρ can be any formula in our logic,

[5] Note that '$\Rrightarrow$' does not represent logical implication and is not formally part of our language.

although the semantics ensure that the rule can only fire if the condition λ can be evaluated without the use of information from future states[6]. The expectation ρ can be a formula expressing desired properties of the states up to the present and/or a constraint on the future sequence of states that should be monitored for fulfilment or violation.

A formula $\mathsf{Exp}(\lambda, \rho, n, \phi)$ means that the formula ϕ is an active expectation as a result of the rule $\lambda \Rrightarrow \rho$ having fired (i.e. its condition λ becoming true) in a (possibly prior) state specified by nominal n. If the rule fired in a prior state, but the expectation was not immediately fulfilled or violated, then the current form of the expectation ϕ may be different from the expectation ρ in the rule due to the use of *formula progression* (explained below) to carry forward an expectation from one state to the next.

The operators $\mathsf{Fulf}(\lambda, \rho, n, \phi)$ and $\mathsf{Viol}(\lambda, \rho, n, \phi)$ have the same argument structure as Exp, and mean that the rule $\lambda \Rrightarrow \rho$ firing in the state specified by n has resulted in an active expectation ϕ that is fulfilled or (respectively) violated in the current state. These three operators are defined as follows (where n is a nominal):

$$\mathsf{Exp}(\lambda, \rho, n, \phi) \iff (n \wedge \mathsf{Trunc_S}\ \lambda \wedge \phi{=}\rho) \vee \exists\psi{\ominus}(\mathsf{Exp}(\lambda, \rho, n, \psi) \wedge \neg\mathsf{Trunc_S}\ \psi \wedge \neg\mathsf{Trunc_S}\ \neg\psi \wedge \mathsf{Progress}(\psi, \phi))$$

$$\mathsf{Fulf}(\lambda, \rho, n, \phi) \iff \mathsf{Exp}(\lambda, \rho, n, \phi) \wedge \mathsf{Trunc_S}\ \phi$$

$$\mathsf{Viol}(\lambda, \rho, n, \phi) \iff \mathsf{Exp}(\lambda, \rho, n, \phi) \wedge \mathsf{Trunc_S}\ \neg\phi$$

The definition of Exp states that there are two ways for an expectation to result from a rule $\lambda \Rrightarrow \rho$: either λ holds in the current state (without recourse to future information) and therefore ρ is now expected (i.e. it is an expected constraint on the model), or some other formula ψ was expected in the previous state as a result of the rule, ψ was not known to be true or false in that state given the model up to that point, and thus a "progressed" form of ψ is now expected. Progress is a temporal operator corresponding to the progression function defined by Bacchus and Kabanza [4] for planning with "temporally extended goals". Details are beyond the scope of this paper, but essentially, progression transforms a temporal formula from the viewpoint of one state into the viewpoint of the next state. A formula that can be determined to be true (respectively false) without recourse to any future states progresses to $\top$ (respectively $\bot$). A formula that requires future information in order to be fully evaluated is partially evaluated using information from the model up to the current state and is then re-expressed as an equivalent constraint in the context of the next state. For example, if p holds in the current state, then $p \wedge \bigcirc q$ progresses to q, expressed as $\mathsf{Progress}(p \wedge \bigcirc q, q)$.

The Exp, Fulf and Viol operators defined above are rather specific in the information they express about a currently active, fulfilled or violated expectation: the third and fourth arguments record the state in which the rule's condition became true and the current form of the expectation. In many cases, it may be sufficient to know there is currently an active, fulfilled or violated expectation resulting from a given rule. We therefore overload these operators and define alternative versions in which the last two arguments are omitted due to an implicit existential quantification:

[6] Future states might be available in offline monitoring of expectations, such as the examination of an audit trail.

$$\mathsf{Exp}(\lambda, \rho) \iff \exists n,\phi\, \mathsf{Exp}(\lambda, \rho, n, \phi)$$
$$\mathsf{Fulf}(\lambda, \rho) \iff \exists n,\phi\, \mathsf{Fulf}(\lambda, \rho, n, \phi)$$
$$\mathsf{Viol}(\lambda, \rho) \iff \exists n,\phi\, \mathsf{Viol}(\lambda, \rho, n, \phi)$$

Using these operators and the model checker described previously [10] we can now analyse an observed execution trace to check for the activation, fulfilment or violation of expressive temporal rules of expectation, or, as special cases, more restricted representations used in prior work. For example, fulfilment of an obligation $O(\rho \leq \delta)$, stating that condition ρ must be brought about before deadline proposition δ becomes true [9], can be represented as $\mathsf{Fulf}(\top, \neg\delta\, \mathsf{U}\, (\rho \wedge \neg\delta)))$.

Note that we could also introduce additional versions of the Exp, Fulf and Viol operators that existentially quantify over only n or only ϕ; however in the remainder of this paper we will focus on the two-argument versions of the operators.

3.2 Nesting Expectation Operators

In the previous account of the logic, the Exp, Fulf and Viol operators could not contain nested occurrences of these operators. In this paper we allow this nesting, and explain why the previous restriction was unnecessary.

It follows from the definitions given above that the truth of the two versions of the Exp, Fulf and Viol formula do not depend on any future states in the model. This is because they depend only on the truth of a nominal (a special type of proposition) in the current state, Exp and Progress formulae in the prior state, and formulae prefixed by the $\mathsf{Trunc_S}$ operator in the current and prior states. Formula progression, by definition, does not depend on future states, and the $\mathsf{Trunc_S}$ operator eliminates them from consideration. Therefore it is meaningful for Exp, Fulf and Viol operators to appear within a condition of a rule (the first argument) inside one of these operators—the use of the $\mathsf{Trunc_S}$ operator to evaluate rule conditions will work correctly. For example, suppose that a library application has the rule of expectation $book\_borrowed \Rrightarrow \bigcirc book\_returned$ (where each state represents a day). Suppose that we also have a contrary-to-duty rule of the form $\mathsf{Viol}(book\_borrowed, \bigcirc book\_returned) \Rrightarrow fine$, indicating that failure to return a book on time results in a fine (or, more precisely, in the expectation that a fine be imposed). In order to evaluate the formula $\mathsf{Exp}(\mathsf{Viol}(book\_borrowed, \bigcirc book\_returned), fine)$ we need to check whether in the current or any previous state a book was borrowed, and whether this book was returned on time or not. The key point is that in order to evaluate the nested Viol formula, we never need to consult any future timepoints, due to the use of TruncS in the semantics of Exp and Viol.

Additionally, to appear as the expectation of a rule (the second argument) within one of these operators, a formula must be able to be progressed when required—see the second line of the definition of the four-argument Exp operator. The axioms defining the progression operation that were defined previously [10] include the following base

cases[7] (adapted slightly here for simplicity of presentation):

$$\mathcal{M}, i \models \mathsf{Progress}(\phi, \top) \text{ if } \mathcal{M}^i, i \models^+ \phi$$
$$\mathcal{M}, i \models \mathsf{Progress}(\phi, \bot) \text{ if } \mathcal{M}^i, i \models^+ \neg\phi$$

where $\mathcal{M}$ is a model, i is the index of a state in the model, and $\mathcal{M}^i$ denotes the model with all states after index i removed.

As Exp, Fulf and Viol can be evaluated without using any future states in the model, then one of the two base cases above will apply, and formulae having these operators as their principal functor will progress to either $\top$ or $\bot$. Therefore, these operators can also appear nested within the second arguments of these three types of formula and the restriction on nesting Exp, Fulf and Viol imposed in our previous work is unnecessary. Finally, the model checking process described in our earlier work [10] can be easily extended to apply to nested expectations, and we have extended our tool to be able to do so (as we will demonstrate in Section 4.1).

4 Use cases for nested expectation operators

In the previous section we argued that the restriction in previous work which did not allow expectations to be nested was unnecessary, and that the semantics of nested expectations are well defined and unproblematic. We also argued that checking whether nested expectations hold, are fulfilled or are violated, can be easily done within the existing framework and tool [10].

In this section we argue that allowing for nested expectations allows for a range of scenarios to be easily specified. Since we are making the case that nested expectations provide additional expressivity that is useful in a broad range of cases, we provide a number of different use cases in which nested expectations are used to specify desired normative behaviour. Space limitations prevent us from developing each of the scenarios in detail, but the aim is not to provide details on any given case, but, rather, to argue that a wide range of scenarios exists where there is a benefit from allowing nested expectations in a declarative way.

Chained expectations. One use case scenario for nested expectations is to allow for causality relationships between expectations to be captured. In this case, we may want to specify that a certain expectation ω exists when some other expectation has been fulfilled. We can express this as follows:

$$\mathsf{Fulf}(\phi, \psi) \Rrightarrow \omega$$

In other words, once rule $\phi \Rrightarrow \psi$ is fulfilled, ω is expected.
This sequential fulfilment of expectations could arise when the two commitments must be fulfilled in a certain order due to one setting up the conditions for the other to be attempted, or when an agent's responsibilities are escalated as a result of successful performance.

[7] Other axioms for Progress (not shown here) define $\mathsf{Progress}(\phi, \psi)$ compositionally based on the principal functor of ϕ and involve recursive progression of the top-level subformulae of ϕ.

Fulfilment ends probationary period. Another scenario which is complementary is that an expectation ω is *dropped* once another expectation is fulfilled:

$$\top \Rrightarrow \omega \ \mathsf{W}\, \mathsf{Fulf}(\phi, \psi)$$

where W is the "weak until" operator: $\alpha \mathsf{W} \beta \equiv (\alpha \mathsf{U} \beta) \vee \Box \alpha$. In other words, ω is (unconditionally) expected until rule $\phi \Rrightarrow \psi$ is fulfilled, or, if the rule is never fulfilled, it is always expected.
This encodes the situation where some condition applies (e.g. limited access to resources) until an agent ends a probationary period by fulfilling a certain expectation (such as passing a test).

"Contrary to duty" expectation or expectation to act on violation. Whereas the previous two cases dealt with the fulfilment of expectations, and how fulfilment may specify the termination or creation of another expectation, this rule deals with violation, and how it may result in the creation of another expectation:

$$\mathsf{Viol}(\phi, \psi) \Rrightarrow \omega$$

In other words, when rule $\phi \Rrightarrow \psi$ is *violated*, ω is expected.
This type of rule represents the well known concept of a contrary to duty expectation: if one expectation is violated an alternative expectation is created. If ω involves a different agent to the one that violated the first expectation, this can represent the requirement for that agent to respond to the violation. A concrete example of this case that we discussed earlier is the expectation that a fine be imposed should a library book not be returned on time.

Expectation handling priority. The next few scenarios show how nested expectations can be used to specify constraints on the *timing* of expectations. For instance, the following rule expresses a priority between two expectations:

$$\mathsf{Fulf}(\phi, \psi) \Rrightarrow \mathsf{Fulf}(\lambda, \rho)$$

In other words, when rule $\phi \Rrightarrow \psi$ is fulfilled, rule $\lambda \Rrightarrow \rho$ should have been fulfilled already. This could be used to express a policy for placing a priority on the order of fulfilment of rules.

Just-in-time expectation management. The following form of rule could be used to encode a policy that resources for fulfilling a given expectation are made available the moment that expectation becomes active.

$$\mathsf{Exp}(\phi, \psi) \Rrightarrow \omega$$

In other words, once rule $\phi \Rrightarrow \psi$ is triggered, ω is expected.

Delaying rule activation. This, and the next example, deal with constraints on the timing of an expectation relative to an arbitrary condition ω. The following rule

expresses the policy to avoid the conditions that trigger an expectation until appropriate resources are in place for fulfilling it ("ω").

$$\top \Rrightarrow (\neg \mathsf{Exp}(\phi, \psi)) \,\mathsf{U}\, \omega$$

In other words, avoid triggering the rule $\phi \Rrightarrow \psi$ until ω is true.

Delaying rule fulfilment. Similar to the previous example, this is a constraint on the timing of an expectation relative to ω, but here we are specifying that the agent should not *fulfil* the expectation until ω. This may be desirable if, for example, an agent has a policy to not be over-diligent in fulfilling an expectation, e.g. it might only pay bills on the last possible day for payment.

$$\top \Rrightarrow (\neg \mathsf{Fulf}(\phi, \psi)) \,\mathsf{W}\, \omega$$

In other words, avoid fulfilling the rule $\phi \Rrightarrow \psi$ until (and only if[8]) ω becomes true.

Avoid violation between two states. Finally, this and the subsequent scenario deal with constraints over a time *interval*. Given two nominals, n_1 and n_2 we can specify that in the interval defined by the two end points a given condition must hold. For example, we may require that within a designated time interval a certain expectation should not be violated:

$$n_1 \Rrightarrow (\neg \mathsf{Viol}(\phi, \psi)) \,\mathsf{U}\, n_2$$

In other words, avoid violating the rule $\phi \Rrightarrow \psi$ between the states referenced by the nominals n_1 (inclusive) and n_2 (exclusive).
This example may be used in situations where an agent may be willing to risk violation of an expectation, but not during certain periods (e.g. when the boss is in the office).

Fulfil a rule sometime between two states. Similarly to the previous scenario, within a given time interval we can specify a condition, in this case that something (such as an expectation being fulfilled) *should* happen:

$$n_1 \Rrightarrow \neg n_2 \,\mathsf{U}\, (\neg n_2 \wedge \mathsf{Fulf}(\phi, \psi))$$

In other words, the rule $\phi \Rrightarrow \psi$ must be fulfilled sometime between the states referenced by the nominals n_1 (inclusive) and n_2 (exclusive). The formalisation can be paraphrased as once n_1 has occurred, n_2 cannot occur until after $\mathsf{Fulf}(\phi, \psi)$.
This example may be used in situations where an agent subject to an expectation may adopt the policy of fulfilling it in a more restricted period than was originally required, e.g. while the boss in the office and able to directly observe the fulfilment.

4.1 Model checking example

In this section we briefly illustrate that the model checker, extended with the ability to progress expectation modalities, behaves as expected. We use the last use case above as

[8] As before, W is the "weak until" operator: $\alpha \mathsf{W} \beta \equiv (\alpha \mathsf{U} \beta) \vee \Box \alpha$.

an example. Consider the rule $p \Rrightarrow \Diamond q$ (once p holds, q is expected to hold eventually). We can encode the property that this rule will be fulfilled using our Python-based model checker as follows, where formulae are written in prefix form as nested tuples.

```
f = Formula(('Fulf', 'p', ('U', True, 'q')))
```

Now consider a model with four states $s_0, \ldots, s_3$ in which p holds in state s_1 and q holds in state s_3. We encode this as follows, where the `4` is the number of states in the model, and the second argument maps each proposition to a list of indices of states in which it holds. For instance, `'p':{1}` indicates that proposition `p` holds in state 1 (i.e. the second state, since the first state has index 0).

```
m = Model(4,{'p':{1}, 'q':{3})
```

After invoking the model checker on this formula we can examine the value of property `f.labels` to find that the fulfilment formula `f` is only satisfied in state s_3 (we simplify the data structure to suppress details of the label structure not relevant to this paper):

```
{0: False, 1: False, 2: False, 3: True}
```

We now modify the formula so that our policy is to only fulfil the original rule from s_1 onwards and before s_3:

```
f = Formula(('Fulf', 's1',
               ('U', ('not', 's3'),
                 ('and', ('not', 's3'),
                   ('Fulf', 'p', ('U', True, 'q'))))))
```

This definition for `f` is simply the Python-based encoding of the formula:

$$\mathsf{Fulf}(s_1, \neg s_3 \,\mathsf{U}\, (\neg s_3 \wedge \mathsf{Fulf}(p, \Diamond q)))$$

which is the last use case scenario ("Fulfil a rule sometime between two states").

Invoking the labelling function again and examining `f.labels` we now find that the formula is false everywhere:

```
{0: False, 1: False, 2: False, 3: False}
```

This is the expected result as the new formula can only possibly be satisfied in states s_1 and s_2, and the original formula does not hold in those states.

5 Conclusion

In this paper we have considered the ability of approaches for modelling and monitoring techniques various sorts of expectations to represent the existence, fulfilment and violation of expectations as first-class entities, and to allow these to appear nested within rules of expectation. Having found no work in the literature that fully meets these needs, we demonstrated how our existing approach to modelling and monitoring expectations extends readily to address this issue. We listed some use cases to show that the expectations expressible using this new modelling ability are of interest in practical settings.

At present our focus is on passive detection of expectation creation, fulfilment and violation (i.e. monitoring). However, as our use case included several examples of policies that agents might adopt, we need to investigate ways in which agents informed by these expectations can understand them and proactively adjust their behaviour to fulfil such policies. Specifically, an agent could use the model checker to reason about hypothetical extensions of the history so far. Given a history H, and an agent who is considering either action A_1 (resulting in state S_1) or action A_2 (resulting in state S_2), then we could use the model checker to label the extended history $H \oplus S_1$ (where "$\oplus$" denotes sequence concatenation) and the extended history $H \oplus S_2$, and use the results to guide decision making by the agent. More generally, an agent might realise via analysing traces that its current expectation doesn't meet some soft constraint (e.g. getting praised by the boss), and therefore follow a heuristic (or apply some reasoning technique) to restrict the period in which it aims to satisfy the expectation (e.g. so that the boss will witness it). To allow this we must also extend our framework to allow the content of expectations to identify which agents are responsible for particular expectations. Finally, the fulfilment operator (and other operators) could be extended to allow not just a single rule to be given as an argument, but a *set* of rules.

References

1. Alberti, M., Gavanelli, M., Lamma, E., Chesani, F., Mello, P., Torroni, P.: Compliance verification of agent interaction: a logic-based software tool. Applied Artificial Intelligence 20(2), 133–157 (2006)
2. Aldewereld, H., Álvarez-Napagao, S., Dignum, F., Vázquez-Salceda, J.: Making norms concrete. In: Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems. pp. 807–814. IFAAMAS (2010)
3. Artikis, A., Sergot, M.: Executable specification of open multi-agent systems. Logic Journal of the IGPL 18(1), 31–65 (2009)
4. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence 116(1-2), 123–191 (2000)
5. Bentahar, J., Moulin, B., Chaib-draa, B.: Commitment and argument network: a new formalism for agent communication. In: Dignum, F. (ed.) Advances in Agent Communication, LNCS, vol. 2922, pp. 146–165. Springer (2004)
6. Bentahar, J., Moulin, B., Meyer, J.J.C., Lespérance, Y.: A new logical semantics for agent communication. In: Inoue, K., Satoh, K., Toni, F. (eds.) Computational Logic in Multi-Agent Systems, LNCS, vol. 4371, pp. 151–170. Springer (2007)
7. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
8. Boella, G., Torre, L., Verhagen, H.: Introduction to the special issue on normative multiagent systems. Autonomous Agents and Multi-Agent Systems 17(1), 1–10 (2008)
9. Broersen, J., Dignum, F., Dignum, V., , Meyer, J.J.C.: Designing a deontic logic of deadlines. In: Deontic Logic in Computer Science, LNAI, vol. 3065. Springer (2004)
10. Cranefield, S., Winikoff, M.: Verifying social expectations by model checking truncated paths. Journal of Logic and Computation (2010), advance access, doi: 10.1093/logcom/exq055
11. Dignum, F., Meyer, J.J.C., Wieringa, R.: A dynamic logic for reasoning about sub-ideal states. In: ECAI Workshop on Artificial Normative Reasoning. pp. 79–92 (1994)
12. Dignum, F., Weigand, H., Verharen, E.: Meeting the deadline: On the formal specification of temporal deontic constraints. In: Foundations of Intelligent Systems, LNAI, vol. 1079, pp. 243–252. Springer (1996)

13. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Campenhout, D.V.: Reasoning with temporal logic on truncated paths. In: Computer Aided Verification, LNCS, vol. 2725, pp. 27–39. Springer (2003)
14. Esteva, M., Rosell, B., Rodríguez-Aguilar, J.A., Arcos, J.L.: AMELI: An agent-based middleware for electronic institutions. In: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems. vol. 1, pp. 236–243. IEEE Computer Society (2004)
15. Farrell, A.D.H., Sergot, M.J., Sallé, M., Bartolini, C.: Using the event calculus for tracking the normative state of contracts. International Journal of Cooperative Information Systems 14(2 & 3), 99–129 (2005)
16. Fornara, N., Viganò, F., Colombetti, M.: Agent communication and artificial institutions. Autonomous Agents and Multi-Agent Systems 14(2), 121–142 (2007)
17. Gabbay, D.M.: The declarative past and imperative future: Executable temporal logic for interactive systems. In: Temporal Logic in Specification, LNCS, vol. 398, pp. 409–448. Springer (1989)
18. García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A.: Implementing norms in electronic institutions. In: Proceedings of the 4nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005). pp. 667–673. ACM Press (2005)
19. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.W.: Constraint rule-based programming of norms for electronic institutions. Autonomous Agents and Multi-Agent Systems 18(1), 186–217 (2009)
20. Governatori, G., Rotolo, A.: How do agents comply with norms? Dagstuhl Seminar Proceedings 09121, `http://drops.dagstuhl.de/opus/volltexte/2009/1909` (2009)
21. Modgil, S., Faci, N., Meneguzzi, F., Oren, N., Miles, S., Luck, M.: A framework for monitoring agent-based normative systems. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems. pp. 153–160. IFAAMAS, Richland, SC (2009)
22. Singh, M.P.: Semantical considerations on dialectical and practical commitments. In: Cohn, A. (ed.) Proceedings of the 23rd National Conference on Artificial Intelligence. vol. 1, pp. 176–181. AAAI Press (2008)
23. Spoletini, P., Verdicchio, M.: An automata-based monitoring technique for commitment-based multi-agent systems. In: Coordination, Organizations, Institutions and Norms in Agent Systems IV, LNAI, vol. 5428, pp. 172–187. Springer (2009)
24. Vázquez-Salceda, J.: The role of norms and electronic institutions in multi-agent systems applied to complex domains: The harmonia framework. AI Communications 16(3), 209–212 (2003)
25. Vázquez-Salceda, J., Aldewereld, H., Dignum, F.: Implementing norms in multiagent systems. In: Multiagent System Technologies, LNCS, vol. 3187, pp. 313–327. Springer (2004)
26. Verdicchio, M., Colombetti, M.: Communication languages for multiagent systems. Computational Intelligence 25(2), 136–159 (2009)
27. Yolum, P., Singh, M.P.: Commitment machines. In: Meyer, J.J.C., Tambe, M. (eds.) Intelligent Agents VIII, LNCS, vol. 2333, pp. 235–247. Springer (2002)