Verifying social expectations by model checking truncated paths

Stephen Cranefield[1] and Michael Winikoff[2]

[1] Department of Information Science
University of Otago
Dunedin, New Zealand
scranefield@infoscience.otago.ac.nz
[2] School of Computer Science and Information Technology
RMIT University
Melbourne, Australia
michael.winikoff@rmit.edu.au

Abstract. One approach to moderating the expected behaviour of agents in open societies is the use of explicit languages for defining norms, conditional commitments and/or social expectations, together with infrastructure supporting conformance checking. This paper presents a logical account of the fulfilment and violation of social expectations modelled as conditional rules over a hybrid linear propositional temporal logic. Our semantics captures the intuition that the fulfilment or violation of an expectation must be determined without recourse to information from later states. We define a means of updating expectations from one state to the next based on formula progression, and show how conformance checking was implemented by extending the MCLITE and MCFULL algorithms of the Hybrid Logics Model Checker.

1 Introduction

An *electronic institution* [1] is an explicit model of the rules, or norms, that govern the operation of an open multi agent system. A given electronic institution provides rules that agents participating in the institution are expected to follow. These rules can include more traditional protocols (e.g. a request message comes first, followed by either an accept or a refuse), as well as properties that are expected to apply to complete interactions, for example, the norm that any accepted request must be eventually fulfilled.

Since electronic institutions are open systems it is not possible to assume any control over agents, nor is it reasonable to assume that all agents will follow the rules applying to an interaction. Instead, the behaviour of participating agents needs to be monitored and checked, with violations being detected and responded to in a suitable way, such as "punishing" the agent by applying sanctions, or reducing the agent's reputation.

For example, one could specify the behaviour that is expected of agents using rules such as "Once payment is made, the service-providing agent is committed to sending a report to the customer once a week for 52 weeks or until the customer cancels the order"; formalised in a suitable logic [2].

There is therefore a need for mechanisms to check for the fulfilment or violation of norms with respect to a (possibly partial) execution trace. Furthermore, such a mechanism can also be useful for rules of social interaction that are less authoritative than centrally established norms, e.g. conditional rules of expectation that an agent has established as its personal norms, or rules expressing learned regularities in the patterns of other agents' behaviour.

Thus in this paper we focus on modelling the general concept of *social expectation* and demonstrate the use of *model checking* for detecting the fulfilment or violation of such expectations by extending the MCLITE and MCFULL algorithms of the Hybrid Logics Model Checker [3]. The advantages of building on model checking, rather than implementing our own checking algorithm (as was done previously [2]) are that we are working within a clearly defined and well studied verification framework, and that it allows us to extend existing software including a range of optimisations that have been developed for model checking. Although the problem of model checking, in its full generality, is more complex than we need, the problem of model checking a path (a finite or ultimately periodic sequence of states) has also been studied and "can usually be solved efficiently, and profit from specialized algorithms" [4]. We have therefore investigated the applicability of model checking as a way of checking for expectations, fulfilments and violations over a model that is a linear history of observed states.

The theory underlying our approach is designed to apply equally well to both *online* and *offline* monitoring of expectations. For online monitoring, each state is added to the end of the history as it occurs, and the monitoring algorithm works incrementally. The underlying formalism can assume that expectations are always considered at the last state in the history. In contrast, in the offline mode, expectations in previous states are also checked. At each past state, the then-active expectations must be checked for fulfilment without recourse to information from later states: the truth of a future-oriented temporal proposition ϕ at state s over the full history does not imply the fulfilment at s of an expectation with content ϕ.

This paper is structured as follows. Section 2 outlines our intuitions about expectations, fulfilment and violation and sketches out our logical account of these concepts. Section 3 describes the logic and semantic mechanisms we use to express fulfilment and violation of an expectation. In Section 4 we give a brief description of formula progression, a technique used to express the evolution of an unfulfilled and non-violated expectation from one state to the next. Section 5 then describes the Hybrid Logics Model Checker that we have used in this work and the extensions we have made to it. Example output from the extended model checker for two example scenarios is presented in Section 6. Finally we discuss related work in Section 7 and summarise the paper and plans for future work in Section 8.

2 Formalising expectations, fulfilment and violation

In this work we study the general notion of *expectations*. It is our position that the base-level semantics of expectations with different degrees of force (expectations inferred from experience, promises, formal commitments, etc.) are the same. The differences between these lie in the pragmatics of how they are created and propagated, how their

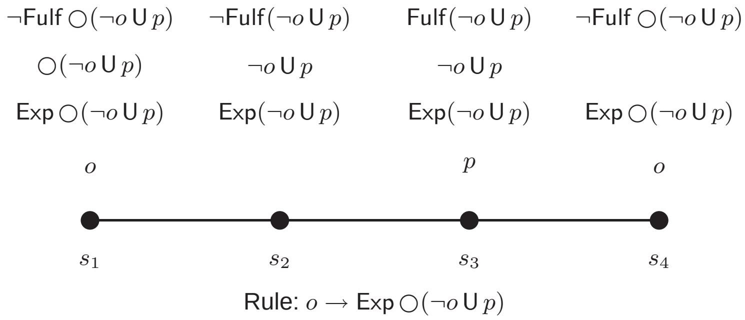

Fig. 1. An example rule and scenario

fulfilment and violation is handled, and the type of contextual information associated with them (e.g. the debtor and creditors associated with a commitment).

Our intuition behind expectations is that they are created in some context which may depend on the current and recorded past states of an agent (including any representation it has of the external environment), and that the created expectation is a constraint indicating the expected future sequences of states. We model this by conditional rules:

$$\lambda \rightarrow \mathsf{Exp}\,\rho$$

where λ and ρ are linear temporal logic expressions with λ referring to the past and present and ρ encoding the constraint on the future. The modality Exp is needed as it is not guaranteed that ρ will hold; it will just be "expected" if the condition holds.

The question then arises of when an expectation should be considered to be fulfilled (denoted $\mathsf{Fulf}(\phi)$) or violated ($\mathsf{Viol}(\phi)$). Consider Fig. 1. This shows a rule expressing an expectation on the interaction between a merchant and a customer: that a customer agent that has placed an order (modelled as the proposition o) should not subsequently place another order until its order has been paid for (proposition p). We formalise this as $o \rightarrow \mathsf{Exp}\bigcirc(\neg o \,\mathsf{U}\, p)$, i.e., when o holds, it is expected that, from the next state on, o is false until p holds. In the figure, the bottom row of formulae show the propositions that are observed in a segment of one possible history: o holds in states s_1 and s_4. and p holds in s_3. The row above this shows the expectations that are *created* by the rule (in states s_1 and s_4) and then *updated* from one state to the next (in states s_2 and s_3, using a technique discussed below). Above this we show the content formulae of these expectations for the first three states, which can easily be seen to hold due to the semantics of the temporal operators $\bigcirc$ and U (a longer segment is needed to evaluate the content of the expectation in s_4). However, as indicated in the top row, we should *not* be able to conclude that these expectations are fulfilled, just because their content formulae are true. The determination of fulfilment and violation must be made *without recourse to future information*. Thus, only in state s_3, when payment is made (p holds), should it be concluded that the current expectation is fulfilled. Section 3 presents a temporal operator $\mathsf{Trunc}_\mathsf{S}$ that allows us to express this restriction to past and present information. For now, we will assume that we have suitable definitions of $\mathsf{Fulf}(\phi)$ and $\mathsf{Viol}(\phi)$, and move on to consider how expectations evolve from one state to the next.

We assume that an expectation can be fulfilled or violated at most once and that an expectation that is not fulfilled or violated in a state should persist (in a possibly modified form) to the next state:

$$\mathsf{Exp}\,\phi \wedge \neg\,\mathsf{Fulf}\,\phi \wedge \neg\,\mathsf{Viol}\,\phi \rightarrow \bigcirc\,\mathsf{Exp}\,\psi$$

What should ψ be? Although at least one alternative approach exists (see Section 7), we believe that the most intuitive representation of an expectation is for it to be expressed in terms of the current state. Thus ψ should represent a change of viewpoint of the constraint represented by the expectation ϕ from the current state to the next state. This can be seen in Fig. 1 where $\mathsf{Exp}\bigcirc(\neg o\,\mathsf{U}\,p)$ from s_1 becomes $\mathsf{Exp}(\neg o\,\mathsf{U}\,p)$ in s_2 and then remains as $\mathsf{Exp}(\neg o\,\mathsf{U}\,p)$ in s_3 as p was not true in s_2.

The transformation of ϕ into ψ should also take into account any simplification of the expectation due to subformulae of ϕ that were true in the current state. Thus, an expectation $\mathsf{Exp}\,(p\wedge\bigcirc q)$ should become $\mathsf{Exp}\,q$ in the next state if p holds currently. This is precisely the notion of formula progression through a state [5]. Formula progression (which will be explained in more detail in Section 4) allows us to complete our informal characterisation of the evolution of expectations through time:

$$\mathsf{Exp}\,\phi \wedge \neg\,\mathsf{Fulf}\,\phi \wedge \neg\,\mathsf{Viol}\,\phi \wedge \mathsf{Progress}(\phi,\psi) \rightarrow \bigcirc\,\mathsf{Exp}\,\psi$$

This conception of expectation, fulfilment and violation has been implemented in a previous progression-based system using a logic combining future and past temporal operators with the guarded fragment of first order logic, binders and a form of nominal [2]. However, although this system used a logical notation for rules of expectation, the detection of fulfilments and violations and the progression of expectations from one state to the next were handled algorithmically, and there was not a logical account of these notions. This paper provides such a logical account, elaborating on the intuition presented above, and demonstrates how to build semantics corresponding to this intuition into a model checker for detecting expectations and their fulfilment and violation.

3 Formal background

The logic we use to model social expectations is a hybrid temporal logic that is an extension of the one implemented by the Hybrid Logics Model Checker [3]. It is described by the following grammar:

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \bigcirc\phi \mid \ominus\phi \mid \phi_1\,\mathsf{U}\,\phi_2 \mid \phi_1\,\mathsf{S}\,\phi_2 \mid x \mid n \mid @_t\phi \mid {\downarrow}x\phi \mid \mathsf{E}\,\phi$$

where p is a proposition, $\bigcirc$ is the standard temporal "next" operator, $\ominus$ is the standard temporal "previous", U is the standard temporal "until", and S ("since") is a backwards-looking version of until. We assume the propositions include $\top$ (true) and $\bot$ (false), with their usual meanings, and define as abbreviations the derived operators "eventually ϕ" ($\Diamond\phi \equiv true\,\mathsf{U}\,\phi$), and "always ϕ" ($\Box\phi \equiv \neg\Diamond\neg\phi$), and similar

$$
\begin{array}{rll}
\mathcal{M}, g, i \models p & \text{iff} & m_i \in V(p) \\
\mathcal{M}, g, i \models \neg\phi & \text{iff} & \mathcal{M}, g, i \not\models \phi \\
\mathcal{M}, g, i \models \phi_1 \wedge \phi_2 & \text{iff} & \mathcal{M}, g, i \models \phi_1 \text{ and } \mathcal{M}, g, i \models \phi_2 \\
\mathcal{M}, g, i \models \bigcirc\phi & \text{iff} & \mathcal{M}, g, i+1 \models \phi \\
\mathcal{M}, g, i \models \ominus\phi & \text{iff} & \mathcal{M}, g, i-1 \models \phi \\
\mathcal{M}, g, i \models \phi_1 \,\mathsf{U}\, \phi_2 & \text{iff} & \exists k \geq i : \mathcal{M}, g, k \models \phi_2 \text{ and } \forall j \text{ such that } i \leq j < k : \mathcal{M}, g, j \models \phi_1 \\
\mathcal{M}, g, i \models \phi_1 \,\mathsf{S}\, \phi_2 & \text{iff} & \exists k \leq i : \mathcal{M}, g, k \models \phi_2 \text{ and } \forall j \text{ such that } i \geq j > k : \mathcal{M}, g, j \models \phi_1 \\
\mathcal{M}, g, i \models x & \text{iff} & m_i = g(x) \\
\mathcal{M}, g, i \models n & \text{iff} & V(n) = \{m_i\} \\
\mathcal{M}, g, i \models @_t\phi & \text{iff} & \mathcal{M}, g, j \models \phi \text{ where } V(t) = \{m_j\} \text{ if } t \text{ is a nominal} \\
 & & \text{and } m_j = g(t) \text{ if } t \text{ is a state variable.} \\
\mathcal{M}, g, i \models \downarrow\! x\phi & \text{iff} & \mathcal{M}, g[x \mapsto m_i], i \models \phi \\
\mathcal{M}, g, i \models \mathsf{E}\phi & \text{iff} & \text{there exists } j \text{ s.t. } m_j \in \mathcal{M} \text{ and } \mathcal{M}, g, j \models \phi
\end{array}
$$

Fig. 2. Infinite-path semantics of the logic

backwards-looking versions $\diamondminus\phi \equiv true\,\mathsf{S}\,\phi$ and $\boxminus\phi \equiv \neg\diamondminus\neg\phi$. In some literature $\Diamond$ is denoted by **F**, $\Box$ by **G**, $\diamondminus$ by $\mathbf{F}^-$ and $\boxminus$ by $\mathbf{G}^-$.

The remaining cases are standard in hybrid logic [6]: we have so-called *state variables*, with typical element x, which can be bound to nominals, and we have nominals n. A nominal is viewed as a logical proposition that is true in exactly one state, i.e. the state "designated" by the nominal. The operator $@_t\phi$, where t is either a state variable or a nominal, shifts evaluation to the state t and can be read as "ϕ holds in state t". The operator $\downarrow\! x\phi$ binds the state variable x to the current state. Finally, the existential modality $\mathsf{E}\,\phi$ says that there exists a state in which ϕ holds, and its dual is the universal modality A. The use of nominals is important: we rely on each state having a unique label in order to define our Exp modality (see Section 5.3).

The formal semantics for this logic is given in Fig. 2 with respect to a hybrid Kripke structure $\mathcal{M}$, which consists of an infinite sequence of states $\langle m_1, m_2 \ldots\rangle$ and a valuation function V that maps propositions and nominals to the set of states in which they hold, i.e. $\mathcal{M} = \langle\langle m_1, m_2 \ldots\rangle, V\rangle$. We use the index i to refer to state m_i. The function g maps state variables x to states, and we write $g[x \mapsto m_i]$ to denote the function that maps x to m_i and otherwise behaves like g. Note that the rules of Fig. 2 only apply for $i \geq 1$. For $i < 1$ we have $\mathcal{M}, g, i \not\models \phi$.

When evaluating whether an expectation is fulfilled in a state m_i we want to not only determine whether the formula holds, but also whether an agent in state m_i is able to conclude that the formula holds. For example, if p is true in m_2, then even through $\bigcirc p$ holds in m_1, an agent in m_1 would not normally be able to conclude this, since it cannot see into the future.

We deal with this by using a simplified form of the operator $\mathsf{Trunc}_\mathsf{S}$ from Eisner *et al.* [7]. A formula $\mathsf{Trunc}_\mathsf{S}\,\phi$ is true at a given state in a model if and only if ϕ can be

Condition	$\mathcal{M}, g, i \models^{+} \phi$?	$\mathcal{M}, g, i \models^{-} \phi$?
$i > \lvert\mathcal{M}\rvert$	false	true
$i \leq \lvert\mathcal{M}\rvert$ and $\phi = \neg\psi$	iff $\mathcal{M}, g, i \not\models^{-} \psi$	iff $\mathcal{M}, g, i \not\models^{+} \psi$
otherwise	As for $\models$, but substitute $\models^{+}$ or $\models^{-}$ (respectively) for $\models$ in recursive definitions	

Fig. 3. Strong and weak semantics on finite paths

shown to hold without any knowledge of future states. We define this formally as:

$$\mathcal{M}, g, i \models \mathsf{Trunc_S}\, \phi \quad \text{iff} \quad \mathcal{M}^i, g, i \models^{+} \phi$$

where $\models^{+}$ represents the *strong semantics* of Eisner et al. (defined below), and $\mathcal{M}^{\,i}$ is defined as follows. Let $\mathcal{M} = \langle\langle m_1 \ldots m_i \ldots\rangle, V\rangle$. We define $V^{\,i}(p) = V(p) \backslash \{m_{i+1} \ldots\}$, that is, V^i gives the same results as V, but without states m_j for $j > i$. We then define $\mathcal{M}^i = \langle\langle m_1 \ldots m_i\rangle, V^i\rangle$. We write $i > |\mathcal{M}|$ to test for states that have been pruned, i.e. if $i > |\mathcal{M}|$ then there is no m_i in $\mathcal{M}$. We write $i \leq |\mathcal{M}|$ to test for states that do exist, i.e. if $i \leq |\mathcal{M}|$ then $m_i \in M$ (where $\mathcal{M} = \langle M, V\rangle$). We need to use the strong semantics ($\models^{+}$) as the standard semantics is defined over infinite sequences of states and does not provide any way to disregard information from future states. The strong semantics is skeptical: it concludes that $\mathcal{M}, g, i \models^{+} \phi$ only when there is enough evidence so far to definitely conclude that ϕ holds. To define negation, we also need its *weak* counterpart, $\models^{-}$. The weak semantics is generous: it concludes that $\mathcal{M}, g, i \models^{-} \phi$ whenever there is no evidence against ϕ so far.

Fig. 3 defines the strong and weak semantics. Note that the semantics of negation switch between the strong and weak semantics: we can conclude *strongly* (respectively weakly) that $\neg\phi$ holds if and only if we can conclude *weakly* (respectively strongly) that ϕ does not hold.

We can now use the $\mathsf{Trunc_S}$ operator to define fulfilment and violation:

$$\mathsf{Fulf}\, \phi \equiv \mathsf{Exp}\, \phi \wedge \mathsf{Trunc_S}\, \phi$$

$$\mathsf{Viol}\, \phi \equiv \mathsf{Exp}\, \phi \wedge \mathsf{Trunc_S}\, \neg\phi$$

4 Formula Progression

As outlined in Section 2, we use the notion of *formula progression* to describe how an unfulfilled and non-violated expectation evolves from one state to the next. Formula progression was introduced in the TLPlan planner to allow “temporally extended goals” to be used to control the system’s search for a plan. Rather than just describing the desired goal state for the plan to bring about, TLPlan used a linear temporal logic formula to constrain the path of states that could be followed while executing the plan. As planning proceeds, whenever a new action is appended to the end of the plan, this formula must be “progressed” to represent the residual constraint left once planning continues from the state resulting from executing that action.

$$\mathcal{M}, g, i \models \mathsf{Progress}(p, \psi) \text{ where } \begin{cases} \psi = \top & \text{if } p \in V(m_i) \\ \psi = \bot & \text{otherwise} \end{cases}$$

$$\begin{aligned} \mathcal{M}, g, i \models \mathsf{Progress}(\phi_1 \wedge \phi_2, \psi_1 \wedge \psi_2) \ \text{ iff } \ & \mathcal{M}, g, i \models \mathsf{Progress}(\phi_1, \psi_1) \text{ and} \\ & \mathcal{M}, g, i \models \mathsf{Progress}(\phi_2, \psi_2) \end{aligned}$$

$$\mathcal{M}, g, i \models \mathsf{Progress}(\neg\phi, \neg\psi) \ \text{ iff } \ \mathcal{M}, g, i \models \mathsf{Progress}(\phi, \psi)$$

$$\mathcal{M}, g, i \models \mathsf{Progress}(\bigcirc\phi, \phi)$$

$$\begin{aligned} \mathcal{M}, g, i \models \mathsf{Progress}(\phi_1 \mathbin{\mathsf{U}} \phi_2, \psi_2 \vee (\psi_1 \wedge (\phi_1 \mathbin{\mathsf{U}} \phi_2))) \ \text{ iff } \ & \mathcal{M}, g, i \models \mathsf{Progress}(\phi_1, \psi_1) \text{ and} \\ & \mathcal{M}, g, i \models \mathsf{Progress}(\phi_2, \psi_2) \end{aligned}$$

$$\mathcal{M}, g, i \models \mathsf{Progress}(\ominus\phi, \ominus\ominus\phi)$$

$$\mathcal{M}, g, i \models \mathsf{Progress}(\phi_1 \mathbin{\mathsf{S}} \phi_2, \ominus(\phi_1 \mathbin{\mathsf{S}} \phi_2))$$

$$\mathcal{M}, g, i \models \mathsf{Progress}(x, \psi) \ \text{ where } \begin{cases} \psi = \top & \text{if } m_i = g(x) \\ \psi = \bot & \text{otherwise} \end{cases}$$

$$\mathcal{M}, g, i \models \mathsf{Progress}(n, \psi) \ \text{ where } \begin{cases} \psi = \top & \text{if } V(n) = \{m_i\} \\ \psi = \bot & \text{otherwise} \end{cases}$$

$$\begin{aligned} \mathcal{M}, g, i \models \mathsf{Progress}(\downarrow\! x\phi, \psi) \ \text{ iff } \ & \mathcal{M}, g, i \models \mathsf{Progress}(\phi[x/n], \psi) \\ & \text{where } V(n) = \{m_i\} \end{aligned}$$

$$\mathcal{M}, g, i \models \mathsf{Progress}(@_t\phi, @_t\phi)$$

$$\mathcal{M}, g, i \models \mathsf{Progress}(\mathsf{E}\phi, \mathsf{E}\phi)$$

Fig. 4. Recursive evaluation of the progression operator

Bacchus and Kabanza considered progression as a function mapping a formula and state to another formula, and defined this function inductively on the structure of formulae in their logic $\mathcal{LT}$—a first-order version of LTL. They proved the following theorem.

Theorem (Bacchus and Kabanza [5]) *Let $M = \langle w_0, w_1, \ldots \rangle$ be any $\mathcal{LT}$ model. Then, we have for any $\mathcal{LT}$ formula f in which all quantification is bounded, $\langle M, w_i \rangle \models f$ if and only if $\langle M, w_{i+1} \rangle \models Progress(f, w_i)$.*

In this theorem, $Progress(f, w_i)$ is a meta-logical function. We wish to define progression as an operator within the logic, and so adapt the above theorem to provide a *definition* of the modal operator $\mathsf{Progress}(\phi, \psi)$:

$$\mathcal{M}, g, i \models \mathsf{Progress}(\phi, \psi) \ \text{ iff } \ \forall \mathcal{M}' \in \overline{\mathcal{M}}(i),\ \mathcal{M}', g, i \models \phi \iff \mathcal{M}', g, i{+}1 \models \psi$$

where $\overline{\mathcal{M}}(i)$ is the set of all possible infinite models that are extensions of $\mathcal{M}^i$ ($\mathcal{M}$ truncated at i) and which preserve all the nominals in $\mathcal{M}$ (including those at indices past i). Apart from the requirement to agree on nominals, the models of $\overline{\mathcal{M}}(i)$ need not agree with $\mathcal{M}$ on the truth of propositions for state indices $j > i$.

We can then obtain the theorems of Fig. 4, which define an inductive procedure for evaluating progression, in conjunction with the use of Boolean simplification to eliminate $\bot$ and $\top$ as subformulae. This procedure is similar to the function of Bacchus and Kabanza, but extended to account for the hybrid features of our logic. The theorem for the binder operator requires there to be a nominal naming the state m_i ($\phi[x/n]$ denotes subsitution of the nominal n for the free occurrences of x); however, for our model checking application, this can be easily ensured by preprocessing the model to add nominals for states that lack them.

5 Applying model checking to expectation monitoring

Model checking is the problem of determining for a *particular* model of a logical language whether a given formula holds in that model. Thus it differs from logical inference mechanisms which make deductions based on rules that are valid in *all* possible models. This makes model checking more tractable in general than deduction.

Model checking is commonly used for checking that models of dynamic systems, encoded as finite state machines, satisfy properties expressed in a temporal logic. However, model checking is also able to check paths, and we have therefore investigated the applicability of model checking as a way of checking for expectations, fulfilments and violations over a model which is a linear history of observed states. This was done by extending an existing model checker, described in the next section.

5.1 The Hybrid Logics Model Checker

The Hybrid Logics Model Checker (HLMC) [3] implements the MCLITE and MCFULL labelling algorithms of Franceschet and de Rijke [8]. HLMC reads a model encoded in XML and a formula given in a textual notation, and uses the selected labelling algorithm to determine the label, true ($\top$) or false ($\bot$), for the input formula in each state of the model. It then reports to the user all the states in which the formula is true (i.e. it is a *global* model checker).

The two labelling algorithms are defined over a propositional temporal logic with the operators **F** ("some time in the future"), **P** ("some time in the past"), the binary temporal operators **U** (until) and **S** (since), the universal modality **A**, and the following features of hybrid logic: nominals, state variables, the operator $@_t$, and the binding operators $\downarrow x$ and $\exists x$ ("binding x to some state makes the following expression true"). The duals of the modal operators are defined in the usual way. The underlying accessibility relation is assumed to represent "later than" (the transitive closure of the "next state" relation underlying many temporal logics), so the definitions of **F** and **P** in terms of the accessibility relation are equivalent to those of our $\bigcirc$ and $\ominus$, respectively, over a "next state" accessibility relation. Time is not constrained to be linear.

The global model checking problem for any subset of this language that freely combines temporal operators with binders is known to be PSPACE-complete [8]. MCLITE is a bottom-up labelling algorithm for the sublanguage that excludes the two binding operators, and it runs in time $O(k.n.m)$ where k is the length of the formula to be

Nominals and state variables

$$\mathcal{M}, g, s \models a \quad \text{iff} \quad s \in [V, g](a)$$

$$L_{\mathcal{M},g}(a, s) \quad = \quad \begin{cases} \top & \text{if } s \in [V, g](a) \\ \bot & \text{otherwise} \end{cases}$$

Operator $@_t$

$$\mathcal{M}, g, s \models @_t\phi \quad \text{iff} \quad \mathcal{M}, g, s' \models \phi \text{ where } [V, g](t) = \{s'\}$$

$$L_{\mathcal{M},g}(@_t\phi, s) \quad = \quad L_{\mathcal{M},g}(\phi, s') \text{ where } [V, g](t) = \{s'\}$$

Operator $\bigcirc$

$$\mathcal{M}, g, s \models \bigcirc\phi \quad \text{iff} \quad \exists s'(s \prec s' \wedge \mathcal{M}, g, s' \models \phi)$$

$$L_{\mathcal{M},g}(\bigcirc\phi, s) \quad = \quad \bigvee_{s' \succ s} L_{\mathcal{M},g}(\phi, s')$$

Fig. 5. The MCLITE labelling function (partial definition)

checked, n is the number of states in the model, and m is the size of the model's accessibility relation. MCFULL handles the full language, uses polynomial space, and runs in time exponential on the nesting degree of the binders in the formula.

MCLITE works by labelling each subformula of the formula to be checked, for all states in the model, in a bottom-up manner. Fig. 5 shows the semantics of some of the operators supported by HLMC together with the corresponding definition of the label $L_{\mathcal{M},g}(\phi, s)$. The presentation is adapted from that of Franceschet and de Rijke [8] to correspond to the HLMC operators, and to provide a declarative rather than procedural account[3]. We use $[V, g](a)$ as an abbreviation for either the value of $V(a)$ if a is a nominal or $\{g(a)\}$ if a is a state variable. It can be seen that in these cases the labelling function is a straightforward translation from the semantics—a property we have sought to preserve where possible for our extended notion of labels presented in Section 5.2.

The simple bottom-up procedure does not work when binders are included in the language as there will be subformulae containing free state variables, and the values of these depend on the enclosing binding context. Instead, the recursive top-down MCFULL procedure is used. A formula is labelled by first labelling its immediate subformulae recursively, and then applying the appropriate labelling algorithm for the formula's operator. For operators in the MCLITE sublanguage, the MCLITE labelling algorithm is used. When the recursion encounters a formula of the form $\downarrow x\phi_x$, the recursive labelling is performed for *each* binding of x to a state in the model (consider the formula $\mathbf{G} \downarrow x\, @_x p$: labelling this for any given state s requires the truth of the subformula to be known for all bindings of x to states accessible from s).

5.2 Handling $\mathsf{Trunc}_\mathsf{S}$

We have adapted HLMC for checking the fulfilment and violation of expectations. We assume (and verify) that the input model represents a linear path and thus contains a single "next state" accessibility relationship.

[3] We use the notation defined in Section 3, but in this more general non-linear setting, we write $\mathcal{M}, g, s \models \phi$ where s is a state in the model rather than the index of a state in the model, and we denote the next-state relation by $\prec$.

To allow the checking of fulfilment and violation a labelling algorithm for $\mathsf{Trunc}_\mathsf{S}$ was developed. This was complicated by the presence of past-time operators. Consider the label for $\mathsf{Trunc}_\mathsf{S} \ominus\neg\phi$. Based on the definitions of Section 3, we have:

$$\begin{aligned}\mathcal{M}, g, i \models \mathsf{Trunc}_\mathsf{S} \ominus\neg\phi &\iff \mathcal{M}^i, g, i \models^+ \ominus\neg\phi \\ &\iff \mathcal{M}^i, g, i{-}1 \models^+ \neg\phi \\ &\iff \mathcal{M}^i, g, i{-}1 \not\models^- \phi\end{aligned}$$

or equivalently:

$$\begin{aligned}L_{\mathcal{M},g}(\mathsf{Trunc}_\mathsf{S} \ominus\neg\phi, i) &= L^+_{\mathcal{M}^i,g}(\ominus\neg\phi, i) \\ &= L^+_{\mathcal{M}^i,g}(\neg\phi, i{-}1) \\ &= \neg L^-_{\mathcal{M}^i,g}(\phi, i{-}1)\end{aligned}$$

where $L^+_{M,g}$ and $L^-_{M,g}$ denote labelling under the strong and weak semantics, respectively. Thus to label $\mathsf{Trunc}_\mathsf{S} \ominus\neg\phi$ at model index i it is necessary to know the weak semantics label for ϕ at index i–1 when the model is truncated at i. More generally, when labelling a formula ϕ at a model index i it is necessary to store both weak and strong labels with respect to all possible future truncation points: $L^-_{M^j,g}(\phi, i)$ and $L^+_{M^j,g}(\phi, i)$ for $j \geq i$. We therefore define a generalised label for a formula ϕ at model index i as a sequence of pairs of weak and strong labels for each possible truncation point from i to the final state in the model:

$$L_{\mathcal{M},g}(\phi, i) = \left\langle \left(L^-_{\mathcal{M}^j,g}(\phi, i),\ L^+_{\mathcal{M}^j,g}(\phi, i)\right) \mid i \leq j \leq |\mathcal{M}| \right\rangle$$

where $L^-_{\mathcal{M}^j,g}(\phi, i)$ is the value for ϕ at index i in the model under the weak semantics assuming a truncation at index j, and $L^+_{\mathcal{M}^j,g}(\phi, i)$ is the corresponding value under the strong semantics.

We will write the value of a label $L_{\mathcal{M},g}(\phi, i)$ in an abbreviated notation that only lists the pairs of weak and strong values when there has been a change of value since the previous possible truncation point:

$$\langle j_1 : (w_{j_1}, s_{j_1}), \ldots, j_{n-1} : (w_{j_{n-1}}, s_{j_{n-1}})\rangle$$

where $i = j_1 < \cdots < j_{n-1} \leq j_n = |\mathcal{M}|$ and $\forall_{1\leq k\leq n-1} \forall_{j_k < l < j_{k+1}} (w_l = w_{j_k} \wedge s_l = s_{j_k})$.

Conjunction and disjunction apply to generalised labels in a straightforward way, acting element-wise, while negation operates on the arguments and then exchanges the resulting weak and strong values at each truncation point, e.g. $\neg\langle 1 : (\top, \bot), 2 : (\top, \top)\rangle = \langle 1 : (\top, \bot), 2 : (\bot, \bot)\rangle$. When $\wedge$ and $\vee$ are applied to labels $l = \langle i : (w_i, s_i), \ldots\rangle$ and $l' = \langle j : (w'_i, s'_i), \ldots\rangle$ where $i < j$, l' is treated as if it had $i : (\bot, \bot)$ prepended for $\vee$ and $i : (\top, \top)$ prepended for $\wedge$ (and l is treated similarly if $i > j$), i.e. the sequence starting at a later truncation point is padded with true weak and strong labels for truncation points i to $j - 1$. We write indexed conjunctions and disjunctions, e.g. ${}^i\bigwedge_{1\leq k\leq|\mathcal{M}|}$,

Out of bound indices

$$L_{\mathcal{M},g}(\phi, i) = \begin{cases} \langle 1:(\bot,\bot)\rangle & \text{for } i < 1 \\ \langle\rangle & \text{for } i > |\mathcal{M}| \end{cases}$$

Operators $\bigcirc$ and $\ominus$ $(1 \leq i \leq |\mathcal{M}|)$

$$L_{\mathcal{M},g}(\bigcirc\phi, i) = i:(\top,\bot) \bullet L_{\mathcal{M},g}(\phi, i{+}1)$$

where $\bullet$ is the prepend operation

$$L_{\mathcal{M},g}(\ominus\phi, i) = L_{\mathcal{M},g}(\phi, i{-}1) \downharpoonright i$$

where $\sigma \downharpoonright i$ is $\langle j:(w_j, s_j) \in \sigma \mid j \geq i\rangle$

Operators U and S $(1 \leq i \leq |\mathcal{M}|)$

$$L_{\mathcal{M},g}(\phi \,\mathsf{U}\, \psi, i) = {}^{i}\!\!\bigvee_{i \leq k \leq |\mathcal{M}|} \Big((L_{\mathcal{M},g}(\psi, k) \vee (L_{\mathcal{M},g}(\phi, k) \wedge \pi^w_k)) \wedge {}^{k}\!\!\bigwedge_{i \leq j < k} L_{\mathcal{M},g}(\phi, j) \Big)$$

where $\pi^w_k = \langle k:(\top,\bot)\rangle$

$$L_{\mathcal{M},g}(\phi \mathsf{S} \psi, i) = \Big({}^{1}\!\!\bigvee_{1 \leq k \leq i} (L_{\mathcal{M},g}(\psi, k) \wedge {}^{k}\!\!\bigwedge_{k < j \leq i} L_{\mathcal{M},g}(\phi, j)) \Big) \downharpoonright i$$

Fig. 6. Labelling temporal formulae in extended HLMC

with a prefix superscript index i, indicating that the value if there are no conjuncts or disjuncts is $\langle i:(\top,\top)\rangle$ or $\langle i:(\bot,\bot)\rangle$ respectively.

For the HLMC operators that are not future oriented, the declarative specifications of the MCLITE/MCFULL labelling algorithms (as shown in part in Fig. 5) can then be specialised for linear models and applied to these generalised labels. Labels for the temporal operators are computed using the definitions in Fig. 6. We also support the derived operators $\Diamond$, $\Box$, $\diamondminus$ and $\boxminus$ defined in Section 3. The expression $\cdots \vee (L_{\mathcal{M},g}(\phi, k) \wedge \pi^w_k)$ used in the definition of U captures the intuition that under the weak semantics, $\phi \,\mathsf{U}\, \psi$ is satisfied if ϕ always holds (weakly) in the future and ψ never does. The constant label π^w_k is used as a mask to ensure that this disjunct only applies for the weak semantics.

5.3 Defining expectation, fulfilment and violation

We now show how the semantics of expectation, fulfilment and violation can be encoded within the extended HLMC. We elaborate on the intuitive account of these notions given in Section 2. We wish to use the model checker to check for the existence of rule-based conditional expectations, and their fulfilments and violations without requiring the rules of expectation to be hard-coded in the model checker, or integrated into the labelling procedure dynamically. Therefore, we define a *hypothetical* expectation modality $\mathsf{Exp}(\lambda, \rho, n, \phi)$. This means (informally) that if there *were* a rule $\lambda \rightarrow \mathsf{Exp}(\rho)$ then λ would have been strongly true at a previous state named by nominal n, the rule would have fired, and the expectation ρ would have progressed (possibly over multiple intermediate states) to ϕ in the current state. This means that we don't have to hardcode rules into the model checker, or provide a mechanism to read and internalise them. Instead, a rule of interest to the user can be supplied as arguments to an input formula using the $\mathsf{ExistsExp}$ modality. It is defined as follows.

$$\begin{aligned}\mathcal{M}, g, i \models^{\pm} \mathsf{Exp}(\lambda, \rho, n, \psi) \ \text{ iff } & \mathcal{M}, g, i \models^{\pm} \mathsf{Trunc_S}\,\lambda,\ V(n) = \{m_i\} \text{ and } \psi = \rho \\ \text{or } \exists\phi \text{ s.t. } & \mathcal{M}, g, i-1 \models^{\pm} \mathsf{Exp}(\lambda, \rho, n, \phi), \\ & \mathcal{M}, g, i-1 \not\models^{\pm} \mathsf{Trunc_S}\,\phi, \\ & \mathcal{M}, g, i-1 \not\models^{\pm} \mathsf{Trunc_S}\,\neg\phi \ \text{ and} \\ & \mathcal{M}, g, i-1 \models^{\pm} \mathsf{Progress}(\phi, \psi)\end{aligned}$$

where we write $\models^{\pm}$ to indicate that the choice between the weak or strong semantics is immaterial as states at indices greater than i play no role in this definition.

The first conjunct in the definition expresses the case in which the hypothetical rule matches the current state. Note that we use $\mathsf{Trunc_S}$ when evaluating the rule's condition λ to restrict it to present and past information only. The second conjunct expresses the case of progressing a non-fulfilled and non-violated exectation from the previous state. Note that in order to use nominals to name the state at which rules apply, we require that the input model has been annotated with nominals for each state.

We also define hypothetical versions of Fulf and Viol as follows:

$$\mathcal{M}, g, i \models^{\pm} \mathsf{Fulf}(\lambda, \rho, n, \phi) \quad \text{iff} \quad \mathcal{M}, g, i \models^{\pm} \mathsf{Exp}(\lambda, \rho, n, \phi) \text{ and } \mathcal{M}, g, i \models^{\pm} \mathsf{Trunc_S}\,\phi$$

$$\mathcal{M}, g, i \models^{\pm} \mathsf{Viol}(\lambda, \rho, n, \phi) \quad \text{iff} \quad \mathcal{M}, g, i \models^{\pm} \mathsf{Exp}(\lambda, \rho, n, \phi) \text{ and } \mathcal{M}, g, i \models^{\pm} \mathsf{Trunc_S}\,\neg\phi$$

These modalities are not used directly by the model checker. Instead we define the following existential version of Exp:

$$\mathcal{M}, g, i \models^{\pm} \mathsf{ExistsExp}(\lambda, \rho) \ \text{ iff } \ \exists n, \phi \text{ s.t. } \mathcal{M}, g, i \models^{\pm} \mathsf{ExistsExp}(\lambda, \rho, n, \phi)$$

with similar definitions for $\mathsf{ExistsFulf}(\lambda, \rho)$ and $\mathsf{ExistsViol}(\lambda, \rho)$. These correspond to the actual queries that we wish to make to the model checker: "are there any expectations (or fulfilments or violations) for a given rule, at any state in the model?"

To compute labels for these existential modalities, we first compute the following *witness function* $W_{\mathcal{M},g,i}$ iteratively for i increasing from 1 to $|\mathcal{M}|$ (where labels for the subformulae λ and ρ have already been computed due to HLMC's top-down recursive algorithm):

$$\begin{aligned}W_{\mathcal{M},g,i}(\mathsf{ExistsExp}(\lambda, \rho)) = & \left\{\begin{array}{ll}\{(n, \rho)\} & \text{where } V(n) = \{m_i\} \\ & \text{if } \mathcal{M}, g, i \models^{\pm} \mathsf{Trunc_S}\,\lambda \\ \varnothing & \text{otherwise}\end{array}\right\} \cup \\ & \{(n, \psi) \mid \exists\phi.(n, \phi) \in W_{\mathcal{M},g,i-1}(\mathsf{ExistsExp}(\lambda, \rho)), \\ & \quad \mathcal{M}, g, i-1 \not\models^{\pm} \mathsf{Trunc_S}\,\phi, \\ & \quad \mathcal{M}, g, i-1 \not\models^{\pm} \mathsf{Trunc_S}\,\neg\phi \text{ and} \\ & \quad \mathcal{M}, g, i-1 \models^{\pm} \mathsf{Progress}(\phi, \psi)\}\end{aligned}$$

This collects all pairs (n, ϕ) making $\mathsf{Exp}(\lambda, \rho, n, \phi)$ true at i for a given λ and ρ. The corresponding label for this formula at i is then $\langle i : (\bot, \bot) \rangle$ if the witness set is empty, and otherwise $\langle i : (\top, \top) \rangle$.

Witness functions are also defined for $\mathsf{ExistsFulf}(\lambda, \rho)$ and $\mathsf{ExistsViol}(\lambda, \rho)$ by taking the subset of pairs (n, ϕ) in $\mathsf{ExistsExp}(\lambda, \rho)$ for which $\mathsf{Trunc}_\mathsf{S}\ \phi$ strongly holds and $\mathsf{Trunc}_\mathsf{S}\ \neg\phi$ strongly holds, respectively.

Finally, we can use the extended HLMC to check for expectations, violations and fulfilments over a given model by performing the global model checking procedure with an empty intial binding g, for an input formula such as $\mathsf{ExistsExp}(\lambda, \rho)$, $\mathsf{ExistsFulf}(\lambda, \rho)$ or $\mathsf{ExistsViol}(\lambda, \rho)$ where condition λ and expectation ρ correspond to some rule of interest. The model checker will report all witnesses for the input formula for all states. This can be easily generalised to apply to disjunctions of input formulae referring to multiple rules.

Although the witnesses for the $\mathsf{ExistsExp}$ modality could be used to generate labels for Exp for a given rule, we do not currently support the use of Exp to appear within rules, and so cannot handle interdependent expectations.

6 An Example

Consider the scenario shown in Fig. 1 (Scenario 1) and the modified scenario (Scenario 2) in which o is also true in s_2, i.e. a second order is placed before payment for the first has been received. In Scenario 2, the expectation created in s_1 and progressed to s_2 is violated in that state, whereas there is no violation in Scenario 1. The following witness lists are output from the model checker for these scenarios, given the input formulae $\mathsf{ExistsExp}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$, $\mathsf{ExistsFulf}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$ and $\mathsf{ExistsViol}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$. The list of witnesses (pairs) beside each state record the current existing, fulfilled or violated expectations (depending on the input formula), alongside a nominal naming the state in which the expectation was created.

Scenario 1

$\mathsf{ExistsExp}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$	$\mathsf{ExistsFulf}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$	$\mathsf{ExistsViol}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$
s_1: $(s_1, \bigcirc(\neg o\,\mathsf{U}\,p))$	s_1:	s_1:
s_2: $(s_1, \neg o\,\mathsf{U}\,p)$	s_2:	s_2:
s_3: $(s_1, \neg o\,\mathsf{U}\,p)$	s_3: $(s_1, \neg o\,\mathsf{U}\,p)$	s_3:
s_4: $(s_4, \bigcirc(\neg o\,\mathsf{U}\,p))$	s_4:	s_4:

Scenario 2

$\mathsf{ExistsExp}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$	$\mathsf{ExistsFulf}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$	$\mathsf{ExistsViol}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$
s_1: $(s_1, \bigcirc(\neg o\,\mathsf{U}\,p))$	s_1:	s_1:
s_2: $(s_2, \bigcirc(\neg o\,\mathsf{U}\,p)), (s_1, \neg o\,\mathsf{U}\,p)$	s_2:	s_2: $(s_1, \neg o\,\mathsf{U}\,p)$
s_3: $(s_2, \neg o\,\mathsf{U}\,p)$	s_3: $(s_2, \neg o\,\mathsf{U}\,p)$	s_3:
s_4: $(s_4, \bigcirc(\neg o\,\mathsf{U}\,p))$	s_4:	s_4:

7 Related Work

There have been a variety of approaches to modelling expectations and commitments formally, some of which are outlined below.

The SOCS-SI system [9] represents conditional expectations as rules with an E modality in their conclusion. Abductive inference is used to generate expectations and these are monitored at run time. The temporal aspects of expectations are restricted to constraint logic programming constraints relating variables representing the time of events.

Verdicchio and Colombetti [10] use a first order variant of CTL* with past-time operators to provide axioms defining the lifecycle of commitments in terms of primitives representing the existence, the fulfilment, and the violation of a commitment in a state. In their approach, commitments are always expressed from the viewpoint of the state in which they were created, and the formula $Comm(e, a, b, u)$, recording that event e created a commitment from a to b that u holds, remains true in exactly that form from one state to the next. Fulfilment is then defined by a temporal formula that searches back in time for the event that created the commitment, and then evaluates the content u at that prior state, for all paths passing through the current state.

Bentahar et al. [11] present a logical model for commitments based on a branching time temporal logic in the context of semantics for argumentation. The semantics use accessibility relations for different types of commitments. These encode deadlines that are associated with commitments on their creation.

Model checking has been applied to statically verifying properties of closed systems of agents (thus their programs are available to form the input model) and also for checking desired properties of institution specifications. A recent example of the latter is the work of Viganò and Colombetti [12]. Of more relevance to this paper is the application of model checking to run-time compliance checking based on observed traces.

Endriss [13] discussed the use of *generalised model checking* for deciding whether a trace of an agent dialogue conforms to a protocol expressed in propositional linear temporal logic.

Spoletini and Verdicchio [14] addressed the online monitoring of commitments expressed in a propositional temporal logic with both past and future operators. They proposed a distributed processing architecture that included formula analyser modules based on alternating automata. The discussion is procedural rather than declarative, so it is difficult to compare the technique to our approach.

8 Conclusions and Future Work

This paper has presented a logical account of the notions of conditional expectation, fulfilment and violation in terms of a linear temporal logic. For offline monitoring of expectations, the problem of determining fulfilment and violation of expectations without recourse to future information was identified as a key problem, and a solution was presented in terms of path truncation and the strong semantics of Eisner et al. [7]. It was then shown how the MCLITE and MCFULL model checking algorithms can be modified to support the truncation operator by using generalised labels that record for a model state the truth values under both the weak and strong semantics for all possible future states. An existing model checker (HLMC) has been modified using these techniques to allow the existence of expectations, and fulfilments and violations of these expectations to be detected.

A hybrid propositional temporal logic was used in this work as that is what was implemented by HLMC. Including nominals allowed our Exp modality to record the states in which expectations were created. However, with our focus on *linear* models, the other hybrid constructs have limited value for defining conditional expectations. We plan to extend our approach to apply to a real-time temporal logic interpreted over timed paths. In this case, binders and state variables become useful for expressing timing relations between states.

We also plan to investigate extending the technique to apply to some suitably constrained fragment of first order temporal logic (e.g. the guarded fragment). Other future work includes modifying the internal data structures and labelling algorithms to support incremental online monitoring of expectations, a detailed analysis of the complexity of the modified algorithms and empirical evaluations.

References

1. Cortés, U.: Electronic institutions and agents. AgentLink News **15** (September 2004) 14–15
2. Cranefield, S.: A rule language for modelling and monitoring social expectations in multi-agent systems. In: Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems. Volume 3913 of Lecture Notes in Computer Science. Springer (2006) 246–258
3. Dragone, L.: Hybrid logics model checker. http://luigidragone.com/hlmc/ (2005)
4. Markey, N., Schnoebelen, P.: Model checking a path. In: CONCUR 2003 – Concurrency Theory. Volume 2761 of Lecture Notes in Computer Science. Springer (2003) 251–265
5. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence **116**(1-2) (2000) 123–191
6. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
7. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Campenhout, D.V.: Reasoning with temporal logic on truncated paths. In: Computer Aided Verification. Volume 2725 of Lecture Notes in Computer Science. Springer (2003) 27–39
8. Franceschet, M., de Rijke, M.: Model checking hybrid logics (with an application to semistructured data). Journal of Applied Logic **4**(3) (2006) 279–304
9. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Compliance verification of agent interaction: a logic-based software tool. In Trappl, R., ed.: Cybernetics and Systems 2004. Volume II., Austrian Society for Cybernetics Studies (2004) 570–575
10. Verdicchio, M., Colombetti, M.: A logical model of social commitment for agent communication. In: Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003), ACM Press (2003) 528–535
11. Bentahar, J., Moulin, B., Meyer, J.J.C., Chaib-draa, B.: A logical model for commitment and argument network for agent communication. In: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), IEEE Computer Society (2004) 792–799
12. Viganò, F., Colombetti, M.: Symbolic model checking of institutions. In: ICEC '07: Proceedings of the ninth international conference on Electronic commerce, ACM Press (2007) 35–44
13. Endriss, U.: Temporal logics for representing agent communication protocols. In: Agent Communication II. Volume 3859 of Lecture Notes in Computer Science. Springer (2006) 15–29
14. Spoletini, P., Verdicchio, M.: Commitment monitoring in a multiagent system. In: Proceedings of the 5th International Central and Eastern European Conference on Multi-Agent Systems. Volume 4696 of Lecture Notes in Artificial Intelligence., Springer (2007) 83–92