Verifying social expectations by model checking truncated paths

Stephen Cranefield
Department of Information Science
University of Otago
Dunedin, New Zealand
scranefield@infoscience.otago.ac.nz

Michael Winikoff
School of Computer Science
and Information Technology
RMIT University
Melbourne, Australia
michael.winikoff@rmit.edu.au

ABSTRACT

One approach to moderating the 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 and the identification and possible punishment of anti-social agents. This paper presents a logical account of the creation, fulfilment and violation of social expectations modelled as conditional rules over a hybrid propositional temporal logic.

The semantics are designed to allow model checking over finite histories to be used to check for fulfilment and violation of expectations in both online and offline modes. For online checking, expectations are always considered at the last state in the history, but 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 issue is addressed by defining fulfilment and violation in terms of an extension of Eisner *et al.*'s weak/strong semantics for LTL over truncated paths.

The update of expectations from one state to the next is based on formula progression and the approach has been implemented by extending the MCLITE and MCFULL algorithms of the Hybrid Logic Model Checker.

1. INTRODUCTION

An *electronic institution* [5] 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.

There is therefore a need for mechanisms for checking 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 only personal norms, or rules expressing regularities in the patterns of other agents' behaviour, learned through experience. Thus in this paper we focus on the general concept of modelling *social expectations* and investigate the use of *model checking* for detecting the fulfilment or violation of such expectations. In particular, our approach has been implemented by extending the MCLITE and MCFULL algorithms of the Hybrid Logic Model Checker [7].

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[1].

Our implemented solution currently addresses offline monitoring, but it is designed to be able to be adapted to an incremental online mode as a future development.

This paper is structured as followed. Section 2 outlines our intuitions about expectations, fulfilment and violation and sketches out our logical representation of these concepts. Section 3 describes the logic we use and the semantic mechanisms needed 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. An example conditional expectation is presented in Section 6, together with the corresponding output from the model checker for two input models. 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 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

[1] Of course offline monitoring can be implemented by applying an online algorithm iteratively, but this is not necessarily the most natural approach theoretically or the most efficient approach in practice.

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.

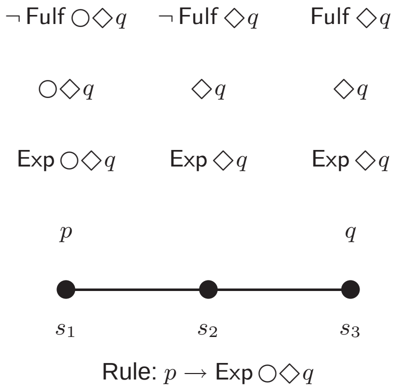
$\neg\,\mathsf{Fulf} \bigcirc\Diamond q$
$\neg\,\mathsf{Fulf} \Diamond q$
$\mathsf{Fulf} \Diamond q$
$\bigcirc\Diamond q$
$\Diamond q$
$\Diamond q$
$\mathsf{Exp} \bigcirc\Diamond q$
$\mathsf{Exp} \Diamond q$
$\mathsf{Exp} \Diamond q$
p
q
s_1
s_2
s_3
Rule: $p \rightarrow \mathsf{Exp} \bigcirc\Diamond q$

Figure 1: Offline monitoring

The question then arises of when an expectation should be considered to be fulfilled (denoted $\mathsf{Fulf}(\phi)$) or violated ($\mathsf{Viol}(\phi)$). Consider Figure 1. This shows a history, as might be used in offline monitoring of agents' interactions (e.g. when analysing a trace presented as evidence by an agent to some authority). In this example there is a rule stating that whenever p holds, it will then be expected that from the next state of the world, q will eventually hold ($\bigcirc$ is the "next state" operator and $\Diamond$ denotes "eventually"). In state s_1, p holds, so the rule is triggered and $\mathsf{Exp} \bigcirc\Diamond q$ holds. Assume that q becomes true in state s_3. Therefore, by the semantics of temporal logic, $\Diamond q$ holds in all three states (not all true formulae are shown in the figure) and, in particular, in s_2. It follows that $\bigcirc\Diamond q$ holds in s_1, the state in which this is an expectatation. However, we do not want to conclude that $\mathsf{Fulf} \bigcirc\Diamond q$ holds in s_1 as an agent in this state would not have access to the future states s_2 and s_3. The determination of fulfilment and violation must be made without recourse to future information. Section 3 presents a temporal operator $\mathsf{Trunc}_\mathsf{S}$ that allows us to express this constraint. 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[2] 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 Figure 1 where $\mathsf{Exp} \bigcirc\Diamond q$ in s_1 becomes $\mathsf{Exp} \Diamond q$ in s_2 and then remains as $\mathsf{Exp} \Diamond q$ in s_3 as q has not yet held.

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, $\mathsf{Exp}\,(p \wedge \bigcirc q)$ should become $\mathsf{Exp}\,q$ in the next state if p holds currently. This is precisely the notion of formula progression through a state [2]. Formula progression (which will be explained in more detail in Section 4) allows us to complete our informal charaterisation 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 expectations, fulfilments and violations has been implemented in a previous progression-based system using a temporal logic combining future and past temporal operators with the guarded fragment of first order logic, binders and a form of nominal [6]. The logic allows the expression of temporally rich conditional expectations 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". However, although this system used a logical notation for conditional 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

In this section we briefly present the logic that we use to express expectations, and its formal semantics. This is followed by introducing a truncation operator, which is defined in terms of weak and strong variants of the semantics. The truncation operator is needed because determining whether an expectation is fulfilled or violated needs to be done without any knowledge of future states, and this is captured by truncating at the current state, so that future states are no longer accessible.

The logic is a hybrid temporal logic that is an extension of the one implemented by the Hybrid Logic Model Checker [7], and is described by the following grammar:

$$\begin{array}{rcl} \phi & ::= & \top \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \\ & & \mid \bigcirc\phi \mid \ominus\phi \mid \phi_1\,\mathsf{U}\,\phi_2 \mid \phi_1\,\mathsf{S}\,\phi_2 \\ & & \mid x \mid n \mid @_t\phi \mid \downarrow\! x\phi \mid \mathsf{E}\,\phi \end{array}$$

where $\top$ represents *true*, 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. As is often done, we define a number of convenient abbreviations: "false" ($\bot \equiv \neg\top$), the derived operators "eventually ϕ" ($\Diamond\phi \equiv \mathit{true}\,\mathsf{U}\,\phi$), and "always ϕ" ($\Box\phi \equiv \neg\Diamond\neg\phi$), and similar backwards-looking versions $\diamondminus\phi \equiv \mathit{true}\,\mathsf{S}\,\phi$ and $\boxminus\phi \equiv \neg\diamondminus\neg\phi$. In some literature $\Diamond$ is denoted by **F**, $\Box$ by **G**, $\diamondminus$ by **F**$^-$ and $\boxminus$ by **G**$^-$.

The remaining cases are standard in hybrid logic [4]: 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 to the state t and can be read as "ϕ holds in state t". The downarrow 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 formal semantics for this logic is given in Figure 2 with respect to a hybrid Kripke structure $\mathcal{M}$ which consists of of an infinite *sequence*[3] of states $\langle m_1, m_2 \ldots\rangle$, and a valuation function

[2] This is true if expectations are treated purely logically, but it may be useful for an agent to be notified of the continued fulfilment or the repeated violation of an expectation expressing that something should always hold.

[3] We are only concerned with a sequence of states, and so build this directly in to the model, rather than by using a relation R which simply creates a sequence indirectly.

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

Figure 2: Semantics of the logic

V which maps propositions and nominals to the set of states in which they hold, i.e. $\mathcal{M} = \langle\langle m_1 \ldots m_n\rangle, V\rangle$. 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 Figure 2 only apply for $i \geq 1$.

When evaluating whether an expectation (or any other formula) holds in a given 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. [8]. A formula $\mathsf{Trunc}_\mathsf{S}\ \phi$ is true at a given state in a model if and only if ϕ can be shown to hold without any knowledge of future states. We define this formally as:

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

where $\models^{+}$ represents the use of the *strong semantics* of Eisner et al. (defined below), and $\mathcal{M}^i$ is defined as follows. Let $\mathcal{M} = \langle\langle m_1 \ldots m_i \ldots m_n\rangle, V\rangle$. We define $V^i(p) = V(p)\backslash\{m_{i+1} \ldots m_n\}$, 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.

Figure 3 defines the weak and strong semantics for the cases where they differ from the standard semantics. For rules for the operators that aren't shown in Figure 3, refer to Figure 2, but replace $\models$ consistently with $\models^{-}$ for the weak semantics and with $\models^{+}$ for the strong semantics. For example the rule for $\bigcirc$ is $\mathcal{M}, g, i \models^{-} \bigcirc\phi \iff \mathcal{M}, g, i+1 \models^{-} \phi$ (and similarly for $\models^{+}$).

$$
\begin{array}{rcl}
\mathcal{M}, g, i \models^{-} p & \iff & i > |\mathcal{M}| \text{ or } m_i \in V(p) \\
\mathcal{M}, g, i \models^{+} \neg\phi & \iff & \mathcal{M}, g, i \not\models^{-} \phi \\
\mathcal{M}, g, i \models^{-} \neg\phi & \iff & \mathcal{M}, g, i \not\models^{+} \phi \\
\mathcal{M}, g, i \models^{+} \bigcirc\phi & \iff & i+1 \leq |\mathcal{M}| \text{ and } \mathcal{M}, g, i+1 \models^{+} \phi \\
\mathcal{M}, g, i \models^{+} x & \iff & i \leq |\mathcal{M}| \text{ and } m_i = g(x) \\
\mathcal{M}, g, i \models^{-} x & \iff & i > |\mathcal{M}| \text{ or } m_i = g(x) \\
\mathcal{M}, g, i \models^{-} n & \iff & i > |\mathcal{M}| \text{ or } V(n) = \{m_i\} \\
\mathcal{M}, g, i \models^{-} @_t\phi & \iff & j > |\mathcal{M}| \text{ or } \mathcal{M}, g, j \models^{-} \phi \\
 & & \text{where } V(t) = \{m_j\} \text{ if } t \text{ is a nominal} \\
 & & \text{and } m_j = g(t) \text{ if } t \text{ is a state variable.}
\end{array}
$$

Figure 3: Weak and Strong semantics

Note that the rules for negation switch between the strong and weak semantics: we can conclude strongly (respectively weakly) that $\neg\phi$ holds iff we can conclude *weakly* (respectively strongly) that ϕ does not hold.

A key intuition in the rules of Figure 3 is that for states that have been pruned, the strong semantics will not conclude that any formula ϕ holds (i.e. for all ϕ we have $\mathcal{M}, g, i \not\models^{+} \phi$ if m_i has been pruned). On the other hand, in this case the weak semantics will conclude that any formula holds (i.e. for all ϕ we have $\mathcal{M}, g, i \models^{-} \phi$ if m_i has been pruned).

The rules for p state that p can be concluded to hold under the strong semantics if the current state $m_i \in V(p)$, recall that when we truncate we update V so it no longer includes states that have been truncated. In the weak semantics p holds either if $m_i \in V(p)$ as in the other semantics, or if m_i has been truncated (i.e. it is not in the sequence of states M).

The $\models^{+}$ rule for $\bigcirc$ requires that the next state must not have been pruned, otherwise we cannot conclude that $\bigcirc\phi$ holds in the strong semantics.

For state variables to hold in the strong semantics we require that $m_i = g(x)$, but also that m_i is in the sequence of states. For state variables to hold in the weak semantics we require that $m_i = g(x)$ *or* that $i > |\mathcal{M}|$. This definition is in line with the key intuition discussed earlier. Similarly, for a nominal to hold in the weak sequence we require that $V(n) = \{m_i\}$ *or* that the current state has been pruned ($i > |\mathcal{M}|$). Similarly, for $@_t$ we allow for this to hold in the weak semantics if the state being shifted to has been pruned.

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

$$\mathsf{Fulf}\,\phi \iff \mathsf{Exp}\,\phi \wedge \mathsf{Trunc}_\mathsf{S}\,\phi$$

$$\mathsf{Viol}\,\phi \iff \mathsf{Exp}\,\phi \wedge \mathsf{Trunc}_\mathsf{S}\,\neg\phi$$

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.

$$
\begin{array}{rcl}
\mathcal{M}, g, i \models \mathsf{Progress}(\top, \top) & & \\
\mathcal{M}, g, i \models \mathsf{Progress}(p, \psi) & \text{where} & \begin{cases} \psi = \top & \text{if } p \in V(m_i) \\ \psi = \bot & \text{otherwise} \end{cases} \\
\mathcal{M}, g, i \models \mathsf{Progress}(\phi_1 \wedge \phi_2, \psi_1 \wedge \psi_2) & \iff & \mathcal{M}, g, i \models \mathsf{Progress}(\phi_1, \psi_1) \text{ and } \mathcal{M}, g, i \models \mathsf{Progress}(\phi_2, \psi_2) \\
\mathcal{M}, g, i \models \mathsf{Progress}(\neg\phi, \neg\psi) & \iff & \mathcal{M}, g, i \models \mathsf{Progress}(\phi, \psi) \\
\mathcal{M}, g, i \models \mathsf{Progress}(\bigcirc\phi, \phi) & & \\
\mathcal{M}, g, i \models \mathsf{Progress}(\phi_1 \,\mathsf{U}\, \phi_2, \psi_2 \vee (\psi_1 \wedge (\phi_1 \,\mathsf{U}\, \phi_2))) & \iff & \mathcal{M}, g, i \models \mathsf{Progress}(\phi_1, \psi_1) \text{ and } \mathcal{M}, g, i \models \mathsf{Progress}(\phi_2, \psi_2) \\
\mathcal{M}, g, i \models \mathsf{Progress}(\ominus\phi, \ominus\ominus\phi) & & \\
\mathcal{M}, g, i \models \mathsf{Progress}(\phi_1 \,\mathsf{S}\, \phi_2, \ominus(\phi_1 \,\mathsf{S}\, \phi_2)) & & \\
\mathcal{M}, g, i \models \mathsf{Progress}(x, \psi) & \text{where} & \begin{cases} \psi = \top & \text{if } m_i = g(x) \\ \psi = \bot & \text{otherwise} \end{cases} \\
\mathcal{M}, g, i \models \mathsf{Progress}(n, \psi) & \text{where} & \begin{cases} \psi = \top & \text{if } V(n) = \{m_i\} \\ \psi = \bot & \text{otherwise} \end{cases} \\
\mathcal{M}, g, i \models \mathsf{Progress}(\downarrow\! x\phi, \psi) & \iff & \mathcal{M}, g, i \models \mathsf{Progress}(\phi[x/n], \psi) \\
 & & \text{where } V(n) = \{m_i\} \\
\mathcal{M}, g, i \models \mathsf{Progress}(@_t\phi, @_t\phi) & & \\
\mathcal{M}, g, i \models \mathsf{Progress}(\mathsf{E}\phi, \mathsf{E}\phi) & &
\end{array}
$$

Figure 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 [2]) *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 $\mathsf{Progress}(\phi, \psi)$:

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

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 Figure 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, 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" [11]. We have therefore investigated the applicability of model checking as a way of checking for expectations, fulfilments and violations over 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) [7] implements the MCLITE and MCFULL labelling algorithms of Franceschet and de Rijke [10]. 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 Tense Logic 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 $@_t$ ("shift evaluation to the state named by nominal or state variable a"), and the binding operators $\downarrow x$ ("bind x to the current state") 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 global model checking problem for any subset of this language that freely combines temporal operators with binders is known to be PSPACE-complete [10]. 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 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 subformulae of the formula to be checked, for all states in the model, in a bottom-up manner. Figure 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 [10] to correspond to the HLMC operators, and to provide a declarative rather than procedural account[4]. We use the notation $[V, g](a)$ 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.

Nominals and state variables

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

Operator $@_t$

$$\begin{array}{rcl} \mathcal{M}, g, s \models @_t\phi & \text{iff} & \mathcal{M}, g, s' \models \phi \text{ where } [V, g](t) = \{s'\} \\ L_{\mathcal{M},g}(@_t\phi, s) & = & L_{\mathcal{M},g}(\phi, s') \text{ where } [V, g](t) = \{s'\} \end{array}$$

Operator $\Diamond_R$

$$\begin{array}{rcl} \mathcal{M}, g, s \models \Diamond_R\phi & \text{iff} & \exists s'(Rss' \wedge \mathcal{M}, g, s' \models \phi) \\ L_{\mathcal{M},g}(\Diamond_R\phi, s) & = & \bigvee_{s' \in R(s)} L_{\mathcal{M},g}(\phi, s') \end{array}$$

Figure 5: The MCLITE labelling function (partial definition)

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 $\downarrow x\phi_x$ or $\exists x\phi_x$, 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).

Franceschet and de Rijke claim that "MCFULL can be viewed as a general model checker for the hybridization of *any* temporal logic" by adding appropriate labelling subprocedures for each modal operator [10], and therefore the model checker HLMC was chosen as the basis for this research. However, HLMC is not a direct implementation of MCLITE and MCFULL. It is not specialised to temporal logic as it allows multiple accessibility relations to appear in the model and makes no assumptions about the structure of these relations. Formulae to be checked can include diamond and box modalities for each accessibility relation in both forward and reverse directions, the existential and universal modalities, and the hybrid logic operators $\downarrow x$ and $@_t$. There is no support for **U** and **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.

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:

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

or equivalently:

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

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 a model index as a sequence of pairs of weak and strong labels for each possible truncation point from i to the final state in the model:

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

$L_{\mathcal{M},g}(\phi, i)$ consists of an element (v^w_j, v^s_j) for each j from i to $|\mathcal{M}|$, where v^w_j is the value for ϕ at index i in the model under the weak semantics assuming a truncation at index j, and v^s_j 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 $\forall_{k:\ 1 \leq k \leq n-1} \forall_{l:\ j_k < l < j_{k+1}} (w_l = w_{j_k} \wedge s_l = s_{j_k})$.

This is also how we store generalised labels in our extended version of HLMC.

Boolean functions apply to generalised labels in a straightforward way, acting element-wise, with negation also exchanging weak and strong values for a given 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. ${}^{i}\!\wedge_{1 \leq k \leq |\mathcal{M}|}$, 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 Figure 5) can then be applied directly to these generalised labels. Labels for the temporal operators are computed using the following definitions in Figure 6 (where we use the operator symbols from Section 3). We also support the derived operators $\Diamond$, $\Box$, ⟠ and $\boxminus$ defined in Section 3[5]. 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.

[4] 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.

[5] Note that our use of $\Diamond$ in this context differs from that used in the original HLMC input language. We use models with a "next step" accessibility relation R, so HLMC's $\Diamond_R$ becomes our $\bigcirc$, and our $\Diamond$ corresponds to the transitive closure of R.

Out of bound indices

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

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

$$\begin{aligned} L_{\mathcal{M},g}(\bigcirc\phi, i) &= i\!:\!(\top,\bot) \bullet L_{\mathcal{M},g}(\phi, i{+}1) \\ &\quad \text{where } \bullet \text{ is the prepend operation} \\ L_{\mathcal{M},g}(\ominus\phi, i) &= L_{\mathcal{M},g}(\phi, i{-}1) \downharpoonleft i \\ &\quad \text{where } \sigma \downharpoonleft i \text{ is } \langle j\!:\!(w_j, s_j) \in \sigma \mid j \geq i\rangle \end{aligned}$$

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

$$\begin{aligned} L_{\mathcal{M},g}(\phi\,\mathsf{U}\,\psi, i) &= \bigvee_{i\leq k\leq|\mathcal{M}|}^{i} \Big((L_{\mathcal{M},g}(\psi,k) \vee (L_{\mathcal{M},g}(\phi,k) \wedge \pi_k^w)) \\ &\qquad\qquad \wedge \bigwedge_{i\leq j<k}^{k} L_{\mathcal{M},g}(\phi,j) \Big) \\ &\quad \text{where } \pi_k^w = \langle k\!:\!(\top,\bot)\rangle \\ L_{\mathcal{M},g}(\phi\mathsf{S}\psi, i) &= \Big(\bigvee_{1\leq k\leq i}^{1} (L_{\mathcal{M},g}(\psi,k) \\ &\qquad\qquad \wedge \bigwedge_{k<j\leq i}^{k} L_{\mathcal{M},g}(\phi,j)) \Big) \downharpoonleft i \end{aligned}$$

Figure 6: Labelling temporal formulae in extended HLMC

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.

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

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 $\mathsf{Trunc}_\mathsf{S}$ 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:

$$\begin{aligned} \mathcal{M}, g, i \models^{\pm} \mathsf{Fulf}(\lambda, \rho, n, \phi) \text{ iff } & \mathcal{M}, g, i \models^{\pm} \mathsf{Exp}(\lambda, \rho, n, \phi) \text{ and} \\ & \mathcal{M}, g, i \models^{\pm} \mathsf{Trunc}_\mathsf{S}\,\phi \\ \mathcal{M}, g, i \models^{\pm} \mathsf{Viol}(\lambda, \rho, n, \phi) \text{ iff } & \mathcal{M}, g, i \models^{\pm} \mathsf{Exp}(\lambda, \rho, n, \phi) \text{ and} \\ & \mathcal{M}, g, i \models^{\pm} \mathsf{Trunc}_\mathsf{S}\,\neg\phi \end{aligned}$$

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

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

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):

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

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 peforming the global model checking procedure with an empty intial binding g, for an input formulae 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 an interaction between a merchant and a customer. One possible expectation in this interaction is 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:

$$o \rightarrow \mathsf{Exp} \bigcirc (\neg o\,\mathsf{U}\,p)$$

That is, when the condition o is true, an expectation is created that, from the next state, o is false until p.

Consider now Scenario 1, illustrated by the example segment of a trace shown below (ignoring for now the o in parentheses). This shows four states labelled by the nominals $s_1, \ldots, s_4$ where o is true in s_1 and s_4, and p is true in s_3. Then in s_1 the condition o is true and hence the expectation $\mathsf{Exp}\bigcirc(\neg o\,\mathsf{U}\,p)$ is created. This expectation is progressed to $\mathsf{Exp}\,\neg o\,\mathsf{U}\,p$ in s_2. Since $\neg o$ holds in s_2 but p does not the expectation is further progressed to $\mathsf{Exp}\,\neg o\,\mathsf{U}\,p$ in s_3. Finally, since p holds in s_3, the expectation is fulfilled in s_3.

Now consider a different scenario (Scenario 2), in which o is also true in s_2 (in brackets in the diagram below). The expectation is progressed to $\mathsf{Exp}\,\neg o\,\mathsf{U}\,p$ in s_2, but because $\neg o$ fails to hold in s_2 the expectation is violated in this state.

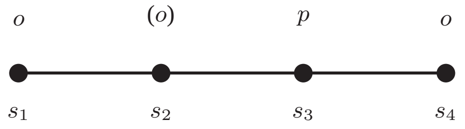

Figure 7 shows the output witness lists from the model checker for these scenarios for the input formulae $\mathsf{ExistsExp}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$, $\mathsf{ExistsFulf}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$ and $\mathsf{ExistsViol}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$.

Scenario 1	**Scenario 2**
$\mathsf{ExistsExp}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$	$\mathsf{ExistsExp}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$
s_1: $(s_1, \bigcirc(\neg o\,\mathsf{U}\,p))$	s_1: $(s_1, \bigcirc(\neg o\,\mathsf{U}\,p))$
s_2: $(s_1, \neg o\,\mathsf{U}\,p)$	s_2: $(s_2, \bigcirc(\neg o\,\mathsf{U}\,p)), (s_1, \neg o\,\mathsf{U}\,p)$
s_3: $(s_1, \neg o\,\mathsf{U}\,p)$	s_3: $(s_2, \neg o\,\mathsf{U}\,p)$
s_4: $(s_4, \bigcirc(\neg o\,\mathsf{U}\,p))$	s_4: $(s_4, \bigcirc(\neg o\,\mathsf{U}\,p))$
$\mathsf{ExistsFulf}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$	$\mathsf{ExistsFulf}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$
s_1:	s_1:
s_2:	s_2:
s_3: $(s_1, \neg o\,\mathsf{U}\,p)$	s_3: $(s_2, \neg o\,\mathsf{U}\,p)$
s_4:	s_4:
$\mathsf{ExistsViol}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$	$\mathsf{ExistsViol}(o, \bigcirc(\neg o\,\mathsf{U}\,p))$
s_1:	s_1:
s_2:	s_2: $(s_1, \neg o\,\mathsf{U}\,p)$
s_3:	s_3:
s_4:	s_4:

Figure 7: Example output from the model checker

In these results, 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.

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 [1] 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.

Verdicchio and Colombetti [13] use a 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.

Yolum and Singh [15] define *commitment machines* as a high-level way of defining agent interaction protocols. The semantics of the language used has an primitive notion of "modal accessibility relations for commitments", but the intuition behind these relations is not explained.

Bentahar et al. [3] also present a logical model for commitments where the semantics of commitments includes accessibility relations for different types of commitments. These encode the deadlines associated with commitments on their creation.

Model checking has been applied to statically verifying properties of institutions as well as interpreting institutions to manage or guide agent interaction, a recent example of the former being the work of Viganò and Colombetti [14]. Of more relevance to this paper is the application of model checking to run-time compliance checking.

Endriss [9] 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 [12] have developed an automata-based approach for online monitoring of the truth values of commitments expressed in a propositional temporal logic with both past and future operators.

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. [8]. 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. This was then used to define fulfilment and violation. 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 histories, 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 complexity of the modified algorithms and empirical evaluations.

9. REFERENCES

[1] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based software tool. In R. Trappl, editor, *Cybernetics and Systems 2004*, volume II, pages 570–575. Austrian Society for Cybernetics Studies, 2004.

[2] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[3] J. Bentahar, B. Moulin, J.-J. C. Meyer, and B. Chaib-draa. 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)*, pages 792–799. IEEE Computer Society, 2004.

[4] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[5] U. Cortés. Electronic institutions and agents. *AgentLink News*, 15:14–15, September 2004.

[6] S. Cranefield. 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*, pages 246–258. Springer, 2006.

[7] L. Dragone. Hybrid logics model checker. http://luigidragone.com/hlmc/, 2005.

[8] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. V. Campenhout. Reasoning with temporal logic on truncated paths. In *Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2003.

[9] U. Endriss. Temporal logics for representing agent communication protocols. In *Agent Communication II*, volume 3859 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006.

[10] M. Franceschet and M. de Rijke. Model checking hybrid logics (with an application to semistructured data). *Journal of Applied Logic*, 4(3):279–304, 2006.

[11] N. Markey and P. Schnoebelen. Model checking a path. In *CONCUR 2003 – Concurrency Theory*, volume 2761 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 2003.

[12] P. Spoletini and M. Verdicchio. 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*, pages 83–92. Springer, 2007.

[13] M. Verdicchio and M. Colombetti. 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)*, pages 528–535. ACM Press, 2003.

[14] F. Viganò and M. Colombetti. Symbolic model checking of institutions. In *ICEC '07: Proceedings of the ninth international conference on Electronic commerce*, pages 35–44. ACM Press, 2007.

[15] P. Yolum and M. P. Singh. Commitment machines. In *Intelligent Agents VIII: 8th International Workshop, ATAL 2001*, volume 2333 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2002.