Modelling and monitoring social expectations in multi-agent systems

Stephen Cranefield

Department of Information Science, University of Otago
PO Box 56, Dunedin, New Zealand
scranefield@infoscience.otago.ac.nz

Abstract. This paper reports on issues confronted and solutions developed while implementing the author's previously proposed hyMITL$^{\pm}$ logic for expressing social expectations as conditional rules. A high level overview of hyMITL$^{\pm}$ is presented, along with a discussion of new features and implementation issues. In particular, the importance of using human-oriented descriptions of time points is argued, along with the need to explicitly take time zones into consideration when defining rules, and a syntax for date/time expressions based on ISO standard 8601 is proposed. A new, more detailed, model for tracking the state of social expectations is also presented, based on the utility of enabling clients of a monitoring service to be notified of multiple instances of the violation or fulfilment of an expectation.

1 Introduction

A significant amount of research in the field of multi-agent systems is currently focused on the theory, design and implementation of *electronic institutions* [1]. This work adapts the mechanisms that keep human society orderly to provide a framework for building open systems of self-interested software agents that are subject to explicitly defined rules of behaviour. Some key requirements in this area are languages for expressing the norms or expectations that apply to agents' interactions and actions, techniques for detecting violations of these rules of society, and mechanisms to prevent or discourage such violations. This paper focuses on the first two requirements, and in particular discusses issues and solutions arising from one previously proposed approach.

The hyMITL$^{\pm}$ logic [2] is a form of temporal logic that allows social expectations to be expressed as rules that are conditional on observations of the past and present, with consequences that impose constraints on future states of the world. The logic and its restricted rule syntax were designed to be amenable to run-time compliance monitoring. This is achieved using an algorithm that keeps a history of observed facts and events, determines when rules are triggered, and applies the technique of *formula progression* [3] to incrementally evaluate and simplify the resulting instantiated consequences (the current expectations) as new states and their associated facts are appended incrementally to the history. When a progressed formula reduces to *true* or *false* this means that a

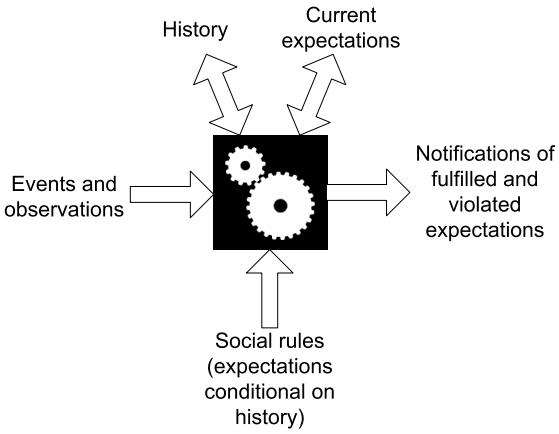

Fig. 1. Overview of the compliance-monitoring process

social expectation has been fulfilled or violated (respectively), and a notification is sent to the clients of the compliance monitor[1]. Figure 1 gives an overview of this process.

The structure of this paper is as follows. A brief overview of hyMITL$^{\pm}$ is given in Section 2, and some implementation choices in maintaining world states for atomic formula evaluation are discussed in Section 3. Section 4 discusses the concept of social expectations as applied to hyMITL$^{\pm}$ and other approaches to modelling electronic institutions. Section 5 advocates the use of a human-oriented time scale in social expectation modelling languages, presents a date/time expression language for hyMITL$^{\pm}$ based on ISO standard 8601, and demonstrates the need for explicit time zone information in date/time expressions involving relative times. The lifecycle of social expectations is discussed in Section 6 and a distinction between local and global compliance is proposed. Finally, some observations are made on the trade-off between the expressiveness of a social expectation modelling language and the types of decision procedure it admits.

2 Overview of hyMITL$^{\pm}$

hyMITL$^{\pm}$ is a temporal logic that includes unary temporal operators (including standard abbreviations) meaning *in the next/previous state* ($\mathsf{X}^{+}/\mathsf{X}^{-}$), *eventually in the future/past* ($\mathsf{F}^{+}/\mathsf{F}^{-}$), and *always in the future/past* ($\mathsf{G}^{+}/\mathsf{G}^{-}$), as well as binary *until* operators for the future and past directions ($\mathsf{U}^{+}/\mathsf{U}^{-}$) and a *for all possible future sequences of states* operator (A). The F, G and U operators are qualified by temporal intervals to constrain the states that must be considered when evaluating the argument (or the second argument in the case of $\mathsf{U}^{+}/\mathsf{U}^{-}$). The default interval (if one is omitted) is

[1] To provide useful information to clients, it is necessary to associate each current expectation with the initial rule instantiation that produced it and the state in which it was fired. This is straightforward, but has not yet been implemented.

$(-\infty, +\infty)$. An interval may also appear on its own as a formula to express the constraint that the current state is within that time period. There is a "current time" binding operator ($\downarrow$) that is qualified by a time unit (e.g. week) and a time zone, and binds the following variable to a term that names the current time in the specified time zone, rounded down to the beginning of the specified unit of time. The following is an example rule expressed using this language, where *p*, *c*, *publication*, *price* and *t* are terms denoting a service provider, a client, a particular publication, a price and a specific date and time, respectively:

$$\mathsf{AG}^{+}\ \mathrm{done}(c,\ \mathrm{buy\_sub}(\mathit{publication}, p, \mathit{price}))\ \wedge\ [t,\ t + \mathsf{P1W}\,|\,\mathsf{Z})\ \rightarrow$$
$$\downarrow_{\mathsf{Z}}^{\mathsf{week}} \mathrm{w}.\mathsf{G}^{+}_{[\mathrm{w}+\,\mathsf{P1W}\,|\,\mathsf{Z},\ \mathrm{w}+\,\mathsf{P53W}\,|\,\mathsf{Z})}$$
$$\downarrow_{\mathsf{Z}}^{\mathsf{week}} \mathrm{cw}.\downarrow_{\mathsf{Z}}^{\mathsf{now}} \mathrm{n}.(\neg \mathsf{X}^{-}[\mathrm{cw}, \mathrm{n}]\ \rightarrow$$
$$\exists \mathrm{d}\ (\mathrm{date\_time\_to\_date}(\mathrm{cw}, \mathrm{d}) \wedge$$
$$\mathsf{F}^{+}_{[\mathrm{n},\ \mathrm{cw}+\,\mathsf{P1W}\,|\,\mathsf{Z})} \mathrm{done}(p,\ \mathrm{send}(c, \mathit{publication}, \mathrm{d}))))$$

This rule states that if client *c* has bought a subscription to *publication* from provider *p* for *price* and this happens within a week of time *t* (the price is only valid for a week), then at all times within the interval beginning a week after the start of the week that the payment is made (w) and ending immediately before 53 weeks after w, if the current state is the first within a given week (encoded as the constraint that the previous state wasn't between the start of the week and now, inclusive), then between now and the end of the week the provider will send the current edition of the publication. In other words, once the payment is made, the publication will be sent every week for 52 weeks.

This example includes some additions and specialisations to the syntax compared to the previous description of hyMITL$^{\pm}$ [2]. In particular, relative times (e.g. P1W) are expressed in a notation based on ISO standard 8601, and time zone annotations (the Zs) are added. These are motivated and discussed in Section 5.

Rules such as this are used by matching the left hand side against a history recording the current and previous states in terms of the events that were observed and the facts that were known to hold in those states[2]. The matching operation results in an instantiation of the right hand side. For example, the rule above will match a state in which the specified *buy_sub* operation has been performed if the date/time associated with that state is less than a week after, or is at, the time t. The resulting instantiated right hand side then represents a social expectation that an expectation monitoring tool can monitor over time. First the new expectation is partially evaluated—in the example this will result in the outer $\downarrow$ operator and its variable being removed and the variable w being instantiated to a term denoting the date/time at which the current week began in the timezone Z. Each time a (relevant) event in the world is observed, a new state is created, atomic formulae describing the observed event and the known facts that hold in that state are asserted into the history, and all expectations being monitored are "progressed" to the new state and partially evaluated, which generally will result in them

[2] The rules may contain past modalities and even future modalities (if nested within past ones), but they should be designed so that the left hand sides can be evaluated using a finite history.

being simplified. If an expectation becomes *true* or *false* on progression, it can be determined that the expectation was fulfilled or violated (respectively). The progression algorithm is not presented formally in this paper, but a high level description is given in Section 6.

3 Representing events and facts

The semantics of hyMITL$^{\pm}$ rely on the standard notion of satisfaction of an atomic formula in a first order model representing a state of the world. When integrating a compliance monitor with an agent platform or institutional middleware, propositions representing observed events and known facts must be asserted into the history of present and past states. While the representation of facts can be based on the ontologies used in the multi-agent system, some convention needs to be adopted for stating that events have just occurred—the example above uses a *done* predicate. A domain model is needed to declare (amongst other things) the event types that are considered relevant to the system and the properties of predicates, e.g. in the example the predicate *date_time_to_date* is used to represent a function that truncates a date/time expression to leave just the date, this can be implemented as a built-in or user-supplied state-independent predicate. The compliance monitor may also need a way of calculating which facts persist from one state to the next, given the events that have just occurred. A mechanism for fact persistence may not be needed if there is middleware that provides an interface for accessing the public institutional state (as in AMELI). However, for more loosely coupled systems the monitor may need to infer the facts that hold in each state based on the facts that held in the previous state and the actions that have occurred. This is precisely the problem that AI planning has addressed with the development of action description formalisms such as STRIPS rules [4] and the situation and event calculi [5, 6]. The domain model could include action descriptions in some existing action description language (such as the event calculus approach of Farrell et al. [7]). Alternatively the hyMITL$^{\pm}$ rule language itself can be used to express this information. Such rules describe the 'physics' or causality of the domain, and their conclusions need to be interpreted not as expectations, but as facts to be asserted into the current state's fact base.

4 Social expectations

Unlike other languages for defining social rules in electronic institutions, hyMITL$^{\pm}$ does not include concepts from deontic logic such as obligation, permission and prohibition, nor does it include any formalised notion of commitments between agents. Including these concepts in a logic allow the fulfilment and violation of norms by agents to be explicitly stated and reasoned about within the language (rather than at the meta-level), as well as allowing the directed social relationships underlying these concepts to be explicitly represented.

While there would be some benefit in adding these features to hyMITL$^{\pm}$, there is also utility in allowing the expression of rules that are not explicitly defined in terms of deontic concepts. The social expectations that an agent has may come from a number of sources. While an electronic insitution will have published rules with official force—in

which case terms such as obligation, permission and prohibition seem appropriate, an agent may usefully maintain its own set of rules expressing social regularities that it has learned, even though these might not have any official status in the institution. Also, rules could be used to express the effects of actions, as discussed above.

In this paper we use the term "social expectations" to encompass any constraints on the present and future that result from rules intended to express social regularities, whether normative or not. We believe that the issues we discuss are relevant to all approaches to modelling and monitoring social expectations.

5 A human-oriented time scale

One of the motivations for the development of MAS technology is to allow humans to decrease their workload or increase their efficiency by delegating work to trusted autonomous software agents (subject to appropriate constraints and policies). Therefore, while some multi-agent systems (e.g. those controlling nuclear reactors or chemical processes) may only need to consider time as a metric quantity measured in (e.g.) milliseconds, many applications of multi-agent systems will require agents to work within human society, and in particular to understand dates and times expressed using human calendar systems. For example, agents may need to understand deadlines expressed in terms of units such as days, weeks and months.

The theory and practicalities of using a human time scale have been addressed to various degrees in the MAS literature. Mallya et al. [8] present example commitments between agents that include relative time expressions such as $t + 7days$, but no syntax and semantics for a date/time language are presented. Verdicchio and Colombetti [9] present a detailed account of the syntax and semantics of date/time expressions and date arithmetic within an agent content language. The normative specification language of Vázquez-Salceda et al. [10] allows the use of absolute and relative deadlines represented in terms of dates and standard time units, but no formal details are presented. Farrell et al. [7] discuss ecXML: a version of the event calculus using an XML syntax, which (based on the examples presented) uses human-oriented date and time units, but this is not explicitly discussed. In our initial presentation of the hyMITL$^{\pm}$ language [2], we showed how a date/time language in the style of Verdicchio and Colombetti can be integrated into a temporal logic in which time intervals and a date/time binding operator are first class elements of the language, rather than being axiomatically defined.

In contrast, other research has treated times as (essentially) real numbers. The ISLANDER e-institution editor [11] and the associated AMELI [12] middleware for governing agents in an institution allow timeouts to be specified in protocol-based norms, and the implementation [13] uses the Java system time in milliseconds for its timestamps. SOCS-SI [14] and the formalism of García-Camino et al. [15] use explicit time variables, arithmetic time expressions, and time inequality constraints, but only numeric time stamps are considered.

In this section we discuss the use of a human-oriented date/time scale in our implementation of the hyHITL$^{\pm}$ logic, in particular, the date/time language used and the qualification of time expressions by time zones.

5.1 A date/time language based on ISO standard 8601

ISO standard 8601 [16] defines standard textual representation formats for dates and times. The defined formats are used (generally in a restricted form, and possibly with some changes) by various Internet and Web standards, such as RFC 3339 [17] for date/time timestamps on the Internet and the XML Schema definition of date and time datatypes [18]. In the implementation of hyMITL$^{\pm}$ we use the formats from ISO 8601 for expressing points in time in terms of date/time units, and for expressing durations in time as *periods*. We also allow new date/time points to be calculated by adding or subtracting relative times to date/time points.

Date/time strings Instances in time are represented using the following syntax:

YYYY`-`*MM*`-`*DD*`T`*hh*`:`*mm*`:`*ss*`.`*fffz*

where *YYYY* is the four-digit number of the year (we assume only AD dates are of interest), and *MM*, *DD*, *hh*, *mm*, *ss* are two-digit representations of the month[3], day, hour (using a 24 hour clock), minute and second, respectively. *fff* represents up to three optional digits for fractions of a second—the preceding decimal point is omitted if there is no fractional part. The `T` separates the date and time components. *z* represents a time zone in terms of an offset to Universal Coordinated Time (UTC). It can be either the character 'Z' (representing the "zero meridian", i.e. an offset of 0), or a '$+$' or '$-$' followed by an hour and minute offset in the form *hh*`:`*mm*.

We assume the Gregorian calendar is used and that the usual constraints on the number of days in each month for a given year are respected.

We do not currently support various abbreviations and variations to this notation allowed by the ISO standard (such as omitting the field separators) or the use of week-of-year or day-of-year expressions.

Period strings An offset in time can be expressed using one of the "period" notations in the standard representing "a duration not associated with any start or end". The notation is:

`P`*years*`Y`*months*`M`*weeks*`W`*days*`DT`*hours*`H`*minutes*`M`*seconds*`S`

where lower case text stands for the desired number of each unit, and the capital letters are unit indicators[4]. Fields and their following unit indicators can be omitted, but the '`T`' separator must be present if there are any time fields. The seconds field can include a decimal point. The leading '`P`' indicates that this is a 'period', and this can be followed by an optional '$+$' or '$-$'.

[3] Unlike the Java Date class, months are numbered from 1.

[4] This format is a slight generalisation of the ISO one as it allows months and weeks to appear together.

Date/time arithmetic We allow expressions denoting the addition or subtraction of periods to date/time points. This is useful when defining date/time points as offsets to date/time variables. The addition of periods to date/time points is complicated as it involves knowledge of the calendar, and it is necessary to have well understood conventions for handling issues such as the variable number of days in a month when adding months to a date and the occurrence of leap years when adding years. Although the ISO standard is not freely available, an algorithm for adding durations to date/time points appears in an appendix of the XML Schema datatypes definition [18]. Our implementation relies on the Joda Time Java library [19] to perform this computation.

A further complication is that the addition of periods to date/time points can only be defined relative to a particular time zone. This issue is discussed in the following section.

5.2 The need for time zones

A period defined in terms of units such as months, weeks and days does not define a fixed length of time. In particular, the addition of months involves an addition to the month component of a date followed by a "rounding down" of the resulting day to an allowed value. This means that the time zone in which the computation is performed can be significant. Consider the following examples, where the subscript to the '+' indicates the timezone used for the addition:

$$\text{2006-02-28T23:00:00Z} +_{\text{Z}} \text{P1M} = \text{2006-03-28T23:00:00Z}$$

$$\text{2006-02-28T23:00:00Z} +_{\text{+01:00}} \text{P1M} = \text{2006-04-01T00:00:00+01:00}$$

This shows that, given the starting date of 11pm, 28 February 2006 (UTC), the addition of a month can result in a difference of three days depending on whether the calculation is performed with respect to UTC or UTC+01:00. To align with people's experience of time, changes to and from summer time must also be reflected in date/time arithmetic.

In the above example, the time zone was provided as a separate annotation to the addition. As an alternative, the timezone associated with the date/time argument could be used ("Z" in both cases above). However, hyMITL$^{\pm}$ can include interval expressions with variables that become instantiated at an outer level of the formula. To ensure that the time zone in which an addition or subtraction is to be performed is explicit in the formula, we use the syntax *date_time* + *period* | *time_zone* as an abbreviation for a ternary addition operator taking an explicit time zone argument. Without this, in the following formula the time zone for the calculation would not be known until the variable *cd* (current day) becomes bound:

$$\begin{aligned}&\text{paid}(\text{cust426}, \text{order77867}) \;\rightarrow \\ &\quad \downarrow_{\text{Z}}^{\text{day}} \text{cd}.\, \mathsf{F}^{+}_{[\text{cd}+\mathsf{P1D}\,|\,\mathsf{Z},\ \text{cd}+\mathsf{P2D}\,|\,\mathsf{Z})} \text{received\_goods}(\text{cust426}, \text{order77867})\end{aligned}$$

This formula states that once a particular customer has paid for a particular order, delivery will be made at some time during the next day. Note that the time binding operator $\downarrow$ must also be qualified by a time zone as well as a time unit.

In practice, for some applications it may be possible to omit time zone annotations and simply use an agent's current time zone. However, in other cases where agents are

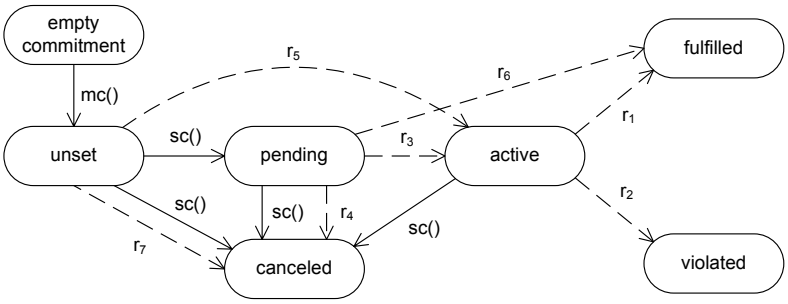

Fig. 2. Fornara and Colombetti's commitment lifecycle [20]

distributed across different time zones it will be crucial to ensure this information is explicitly supplied.

6 The lifecycle of a social expectation

A system that monitors future-oriented social expectations, whether these are obligations, commitments or learned patterns of behaviour, must have some underlying model of the lifecycle of an expectation. For example, various formalisms and practical tools based on commitments have been proposed with differing accounts of the dynamics of a commitment. Figure 2 shows the commitment lifecycle proposed by Fornara and Colombetti [20]. This diagram defines a state space for conditional commitments and the possible transitions between those states, with solid arrows indicating operations (mc for make commitment and sc for set commitment) that occur as a result of agent communication, and dashed arrows indicating state changes that occur as a result of a change in truth value of the commitment's precondition or content propositions. Further constraints on the legal transitions are defined by rules (r_1 to r_7, not shown here) and some "basic authorizations" that restrict the performance of each sc transition to be performed by either the debtor or the creditor of the commitment. The unset state allows an agent to create a commitment for which another agent is the debtor. This can then be set to the pending state by the debtor (if the commitment is accepted), or set to the canceled state (if not accepted).

The *commitment machine* formalism of Yolum and Singh [21] can be viewed as defining a similar state machine, with some additional operations possible on commitments: the release of the commitment by its creditor, the assignment of an alternative agent as the creditor (performed by the original creditor) and the delegation of a commitment by its debtor to an alternative debtor. Because a commitment machine is used to specify protocols in which all commitments are fulfilled, the violation of commitments is not modelled in this formalism. In contrast to the approach of Fornara and Colombetti, a commitment machine does not explicitly represent commitments as propositions with a temporal component—instead, the semantics of commitment assertions directly constrain the possible future paths that conform to a commitment in terms of the satis-

faction of the commitment content in some future state. This is in contrast to the earlier work of Venkatraman and Singh [22] which used the same lifecycle but with particular patterns of CTL formulae as the content of commitments.

In general, a tool to monitor social state will need to track two types of transition in the state of a social expectation: those triggered by interactions between agents (the solid lines in the figure) and those triggered by changes in truth value of the logical content of the expectation (the dashed lines). The ability to monitor the former relies on an ability to overhear communication between agents [23] or the use of group multicasting [24] or a group message redistribution agent [25] when sending messages with important social consequences. Detecting transitions triggered by changes in truth value requires determining whether particular propositions hold or actions have occurred in each state. Vásquez-Salceda et al. [10] have proposed practical implementation techniques for managing this process, and suggested the inclusion of specific detection mechanisms within norm descriptions.

Currently, hyMITL$^{\pm}$ does include any notion of commitments or obligations, and thus a compliance monitor for hyMITL$^{\pm}$ rules is not concerned with monitoring changes of social state. Its focus is on the right hand side of the state diagram in Figure 2. The content of a social expectation having a temporal aspect can have three possible values when an attempt is made to evaluate it. Its value may be unknown (corresponding to the active state in the figure), *true* (the fulfilled state) or *false* (the violated state). As time passes, the compliance monitor's trace of observations and events is extended and expectations with an unknown value may remain in that state or their content may be reduced to a value of *true* or *false*. The hyMITL$^{\pm}$ compliance algorithm presented previously follows this approach using an iterative process of partial evaluation and formula progression [2]. Once a formula has been reduced to *true* or *false*, a fulfilment or violation is reported and the (now trivial) formula is removed from the set of current expectations.

While this may seem an obvious outcome of applying three-valued logic to the evaluation of expectations with a temporal nature, our experience in implementing hyMITL$^{\pm}$ has demonstrated to us that monitoring the transitions between the three states active, fulfilled and violated is not sufficient for compliance monitoring. This is based on the need to allow a wider range of notifications from the compliance monitor to an agent using its services. For example, consider an expectation that an agent will perform a particular operation every day for a year. A client of a compliance monitor tracking this expectation may wish to be notified after every day that the required operation is not performed, not just the first time (which is when the expectation becomes logically false). Any resulting sanctions may depend on the number of repeated violations. The client may also wish to be notified every time the expected action *is* performed, rather than being notified at the end of the year that the expectation as a whole was fulfilled. These examples suggest that a compliance monitor needs distinct notions of *global* versus *local* compliance, and that its clients may wish to control the notifications they receive in a flexible by specifying *notification policies*.

Figure 3 shows a UML 2.0 state machine giving a more detailed account of the possible states of an expectation's content formula, designed to allow more flexible notification to clients of a compliance monitor. The Active state is decomposed into two

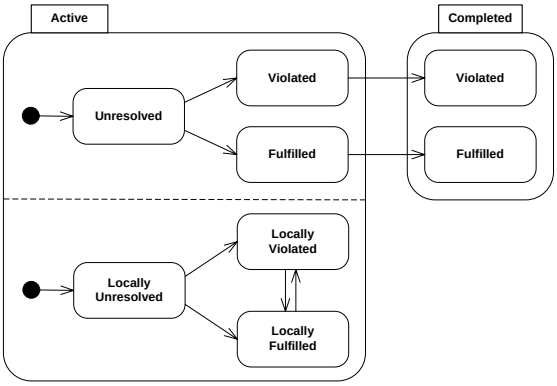

Fig. 3. A more detailed state space for social expectations

orthogonal sets of substates: those indicating the global state, i.e. whether the expectation is logically false, true or unresolved (e.g. if its value depends on the evaluation of future modalities), and those indicating its local state—whether it is true or false when evaluated at the current point in time, ignoring any past violations and requirements on the future. The Completed state represents an expectation that no longer has any relevance, e.g. one that was bounded by a particular time interval that has now passed.

Of course it is still necessary to precisely define the notion of local compliance, and one possible definition follows from the technique of formula progression used in the hyMITL$^{\pm}$ monitoring algorithm. Paraphrasing Kerjean et al. [26]:

> *The idea behind formula progression is to decompose a linear temporal logic formula into a requirement about the present, which can be checked straight away, and a requirement about the (as yet unavailable) next state.*

As hyMITL$^{\pm}$ includes temporal modalities that refer to the past, the compliance monitor keeps a history of past states, and for our purposes Kerjean et al.'s "requirement about the present" becomes a requirement about the present and past. This component can then be evaluated to determine the local compliance of the formula.

The computation is, in fact, a little more complex than the above description suggests. Unlike planning, for which the technique of formula progression was developed, our compliance monitor cannot generate a new state whenever it is ready to progress a formula. It must wait until a new observation is made, which generates a new state. However, it is desirable to deliver any fulfilment or violation notifications about the previous state in a timely fashion. Therefore, we split the progression algorithm into two steps. The first step is a partial evaluation step that recursively evaluates the formula, resolving to *true* or *false* any subformulae that have no future modalities and applying the progression rules to those that do, with any resulting "requirements about the next state" wrapped by the X^{+} operator. It also performs Boolean simplifications. The second step is applied when a new event is observed and the next state is generated. This basically involves removing the outermost X^{+} operators. Given a formula p, the result

of the first step, *peval*(p) determines the local compliance status of the social expectation that this formula is the current value of: if it is *false* the expectation has been locally violated; otherwise (if it is *true* or involves X^+ formulae) the expectation has been locally fulfilled. If the expectation is globally unresolved, then a *peval* result of *true* causes the expectation to become globally fulfilled, and a value of *false* causes it to become globally violated.

A social expectation that has just become globally violated or fulfilled would normally be removed at the next progression step as the current value (*true* or *false*) would have no future-oriented component. However, further local fulfilments or violations can be checked for by progressing the future-oriented part of *peval*(p) before any Boolean simplification is applied. For example, if *peval*(p) evaluates to *true* $\wedge\ \mathsf{X}^+p$, or to *false* $\vee\ \mathsf{X}^+p$, then X^+p could be progressed to the next state, giving p. However, further research is needed to find a general formulation of this idea and a suitably expressive way for clients to specify their desired policies on when this technique should be applied and to what patterns of formula.

7 Expressive power versus inference capability

hyMITL$^{\pm}$ was designed to allow the expression of social rules with complex temporal properties (relative to other approaches), while still being amenable to run-time compliance monitoring. However, the compliance monitoring process is concerned solely with the application of rules and the satisfaction and violation of their consequences, given the history so far. It cannot detect violations of liveness properties, and it does not detect inconsistencies between rules or expectations that are inconsistent, until they have resolved to *true* or *false*. For example, the algorithm will progress both $\mathsf{F}^+_I p$ and $\mathsf{G}^+_I \neg p$, where I is a future interval, until I is reached and one of these formulae is found to be violated. Other approaches to run-time monitoring of expectations have similar limitations [7].

As well as run-time monitoring, there are other decision procedures that may be useful for social expectation modelling languages, e.g.:

- Is a set of rules, or a set of current expectations, consistent?
- Given two sets of rules, which one has the most utility for me?
- What set of rules would ensure that my current goals in society are met?

While there may not be feasible approaches to answering these questions for an expressive language like hyMITL$^{\pm}$, it would be possible to define templates of social contracts that have known properties, with particular parameters that can be varied. Analysis and negotiation could then take place in terms of the parameter space, just as in human society a negotiation over a house purchase usually focuses on the price and occupancy date rather than the fine print of what is often a standard contract.

8 Conclusion

This paper has discussed a number of issues related to the modelling and run-time monitoring of social expectations that have arisen from implementing a monitoring tool

for the hyMITL$^{\pm}$ logic. Further details on this formalism and its implementation have been presented, and in particular a date/time language based on ISO standard 8601 was described, and the use of explicit reference to time zones in such a language was advocated. The lifecycle of social expectations was analysed and a proposal was made for a more detailed account of violation and fulfilment, in order to support a wider range of notifications to clients of a compliance monitor.

Acknowledgements

Thanks to Carles Sierra, Marco Colombetti and Ulises Cortés and their colleagues at IIIA-CSIC, the University of Lugano and Universitat Politècnica de Catalunya (respectively) for their hospitality and thought-provoking discussions during the author's visits in 2005.

References

1. Cortés, U.: Electronic institutions and agents. AgentLink News **15** (2004) 14–15
2. Cranefield, S.: A rule language for modelling and monitoring social expectations in multi-agent systems. In: Selected and Revised papers from the AAMAS 2005 Workshop on Agents, Norms and Institutions for Regulated Multiagent Systems. Volume 3913 of Lecture Notes in Computer Science, Springer (2006) (To appear)
3. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence **116** (2000) 123–191
4. Fikes, R., Nilsson, N.: STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence **2** (1971) 189–208
5. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., Michie, D., eds.: Machine Intelligence. Volume 4. Edinburgh University Press (1969) 463–502
6. Miller, R., Shanahan, M.: The event-calculus in classical logic - alternative axiomatizations. Electronic Transactions on Artificial Intelligence **3** (1999) 77–105
7. Farrell, A.D.H., Sergot, M.J., Sallé, M., Bartolini, C.: Using the event calculus for tracking the normative state of contracts. International Journal of Cooperative Information Systems **14** (2005) 99–129
8. Mallya, A.U., Yolum, P., Singh, M.P.: Resolving commitments among autonomous agents. In: Advances in Agent Communication. Volume 2922 of Lecture Notes in Computer Science, Springer (2004) 166–182
9. Verdicchio, M., Colombetti, M.: Dealing with time in content language expressions. In: Agent Communication: International Workshop on Agent Communication, AC 2004. Volume 3396 of Lecture Notes in Computer Science, Springer (2005) 91–105
10. Vázquez-Salceda, J., Aldewereld, H., Dignum, F.: Implementing norms in multiagent systems. In: Multiagent System Technologies: Second German Conference, MATES 2004. Volume 3187 of Lecture Notes in Computer Science. Springer (2004) 313–327
11. Esteva, M., de la Cruz, D., Sierra, C.: ISLANDER: an electronic institutions editor. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press (2002) 1045–1052
12. Esteva, M., Rosell, B., Rodríguez-Aguilar, J.A., Arcos, J.L.: AMELI: An agent-based middleware for electronic institutions. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems. Volume 1, ACM Press (2004) 236–243

13. IIIA-CSIC: Electronic Institutions Development Environment Web site. http://e-institutions.iiia.csic.es/software.html. Accessed 2006-02-01
14. 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
15. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: A distributed architecture for norm-aware agent societies. In: Declarative Agent Languages and Technologies III: Third International Workshop, DALT 2005. Volume 3904 of Lecture Notes in Computer Science, Springer (2006) 89–105
16. Wikipedia: ISO 8601. http://en.wikipedia.org/wiki/ISO_8601. Accessed 2006-02-01
17. Klyne, G., Newman, C.: Date and time on the internet: Timestamps. Request for Comments 3339, The Internet Society (2002)
18. W3C: XML schema part 2: Datatypes second edition. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/ (2004)
19. Joda.org: Joda Time - Java date and time API. http://joda-time.sourceforge.net/. Accessed 2006-02-1
20. Fornara, N., Colombetti, M.: A commitment-based approach to agent communication. Applied Artificial Intelligence **18** (2004) 853–866
21. Yolum, P., Singh, M.P.: Commitment machines. In: Intelligent Agents VIII: 8th International Workshop, ATAL 2001. Volume 2333 of Lecture Notes in Computer Science, Springer (2002) 235–247
22. Venkatraman, M., Singh, M.P.: Verifying compliance with commitment protocols: Enabling open web-based multiagent systems. Autonomous Agents and Multi-Agent Systems **2** (1999) 217–236
23. Kaminka, G., Pynadath, D., Tambe, M.: Monitoring teams by overhearing: A multi-agent plan-recognition approach. Journal of Artificial Intelligence Research **17** (2002) 83–135
24. Cranefield, S.: Reliable group communication and institutional action in a multi-agent trading scenario. In: Selected and Revised Papers from the AAMAS 2005 Workshop on Agent Communication. Volume 3859 of Lecture Notes in Computer Science. Springer (2006) (To appear)
25. Heard, J., Kremer, R.C.: Practical issues in detecting broken social commitments. In: Selected and Revised Papers from the AAMAS 2005 Workshop on Agent Communication. Volume 3859 of Lecture Notes in Computer Science. Springer (2006) (To appear)
26. Kerjean, S., Kabanza, F., St-Denis, R., Thiébaux, S.: Analyzing LTL model checking techniques for plan synthesis and controller synthesis (work in progress). Electronic Notes in Theoretical Computer Science **149** (2006) 91–104