Temporal Logics for Representing Agent Communication Protocols

Ulle Endriss

Institute for Logic, Language and Computation
University of Amsterdam, 1018 TV Amsterdam, The Netherlands
Email: ulle@illc.uva.nl

Abstract. This paper explores the use of temporal logics in the context of communication protocols for multiagent systems. We concentrate on frameworks where protocols are used to specify the conventions of social interaction, rather than making reference to the mental states of agents. Model checking can be used to check the conformance of a given dialogue between agents to a given protocol expressed in a suitable temporal logic. We begin by showing how simple protocols, such as those typically presented as finite automata, can be specified using a fragment of propositional linear temporal logic. The full logic can also express concepts such as future dialogue obligations (or commitments). Finally, we discuss how an extended temporal logic based on ordered trees can be used to specify nested protocols.

1 Introduction

Communication in multiagent systems is an important and very active area of research [15, 29, 37]. While much work has been devoted to so-called mentalistic models of communication (see in particular [15]), where communicative acts are specified in terms of agents' beliefs and intentions, recently a number of authors have argued for a *convention-based* approach to agent communication languages [9, 23, 29, 31]. Mental attitudes are useful to explain *why* agents may behave in certain ways, but (being non-verifiable for an outside observer) they cannot serve as a basis for specifying the *norms* and *conventions* of interaction required for building open systems that allow for meaningful communication. In the convention-based approach, *protocols* specifying the rules of interaction play a central role.

This paper explores the use of temporal logics in the context of agent communication protocols. Rather than using a form of deontic logic to specify what agents *ought* to do, we use temporal logic formulas to specify the class of all dialogues (sequences of utterances) that are *legal* according to a given protocol. The notion of what an agent ought to do is then implicit: the social conventions of communication are fulfilled, if the generated dialogue satisfies the protocol specification. In particular, we propose to use *propositional linear temporal logic* [16, 18] to specify protocols and *generalised model checking* [7] to decide whether an actual dialogue conforms to such a protocol.

Checking conformance *at runtime*, which is what we are concerned with here, can be distinguished from *a priori* conformance checking which addresses the problem of checking whether an agent can be guaranteed to always conform to a given protocol, on the basis of its specification [12, 19]. Being able to check conformance at runtime is a minimal requirement for systems that operate with a convention-based communication protocol; if violations cannot be detected then the use of such a protocol will be of little use (but how to react to an observed violation is an issue that lies outside the scope of this paper).

The remainder of the paper is structured as follows. Section 2 provides an introduction to agent communication protocols and Section 3 covers the necessary background on temporal logic. In Section 4 the basic ideas of representing dialogues as models, using formulas to specify protocols, and applying (generalised) model checking to verify conformance are introduced. These ideas are then applied to protocols that can be represented as *finite automata* (in Section 5) and to the modelling of *dialogue obligations* (in Section 6). Section 7 discusses ideas on the specification of *nested protocols* using an extended temporal logic based on ordered trees, and Section 8 concludes with a brief discussion of related work.

2 Background on Protocols

An agent communication protocol lays down the conventions (or norms, or rules) of communicative interaction in a multiagent system. Agents communicate with each other by sending messages, which we refer to as *dialogue moves* (or communicative acts, or simply utterances). A *dialogue* is a sequence of such moves. A dialogue move will typically have, at least, the following components: a *sender*, a (list of) *receiver(s)*, a *performative* determining the type of move, and a *content* item defining the actual message content [12, 15, 37]. An example for a performative would be *inform*; an example for a content item would be "the city of Utrecht is more than 1300 years old". Indeed, the *content language* may be highly application-dependent, which means we cannot hope to be able to develop general tools for dealing with this particular aspect of communication. In addition, a dialogue move may also include a time-stamp.

The role of a protocol is to define whether a dialogue is *legal*, *i.e.* whether it conforms to the social rules governing the system to which the protocol in question applies. A variety of mechanisms for the specification of protocols have been put forward in the literature. Pitt and Mamdani [28], for instance, discuss several protocols based on deterministic finite automata. One of these, the *continuous update protocol*, is shown in Figure 1. This protocol may be used to regulate a dialogue where an agent A continuously updates another agent B on the value of some proposition. In each round, B may either acknowledge the information or end the dialogue. Figure 1 only specifies the performative (*inform*, *ack*, or *end*) and the sender (A or B) for each move. In fact, to keep our examples simple, throughout this paper we are going to abstract from the other components of a dialogue move. In the context of automata-based protocols, the definition of legality of a dialogue reduces to the definition of acceptance of a language by an

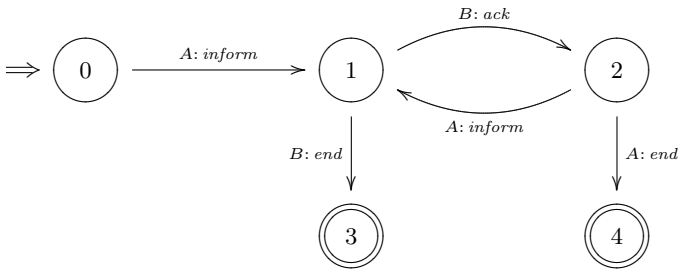


Fig. 1. The continuous update protocol

automaton in the usual sense [26]: A dialogue is legal according to a protocol iff it would be accepted by the automaton corresponding to the protocol.

Protocols defined in terms of finite automata are *complete* in the sense of clearly specifying the range of legal follow-up moves at every stage in a dialogue. This need not be the case, however [1]. In general, any set of rules that put some constraints on a dialogue between agents may be considered a protocol (although complete protocols may often be preferred for practical reasons). Typical examples for protocol rules that constrain a dialogue without necessarily restricting the range of legal follow-ups at every stage are conversational commitments (e.g. to honour a promise) [9], which require an agent to perform a certain communicative act at *some* point in the future. We are going to consider the specification of dialogue obligations like this in Section 6.

3 Background on Temporal Logic

Temporal logic has found many applications in artificial intelligence and computer science. In fact, over the years, a whole family of temporal logics have been developed. In this paper, we are mostly going to use *propositional linear temporal logic* (PLTL), which is probably the most intuitive of the standard temporal logics [16, 18].

We briefly review the syntax and semantics of this logic. The language of PLTL builds on a countable set $\mathcal{L}$ of propositional letters. The set of well-formed *formulas* is the smallest set such that propositional letters are formulas and, whenever φ and ψ are formulas, so are $\neg\varphi$, $\varphi \wedge \psi$, and φ UNTIL ψ. Formulas are evaluated over a *frame* (also known as the *flow of time*). As we are going to identify the points in a frame with the turns in a dialogue (which, for all practical purposes, may be assumed to be finite), we define the semantics of PLTL over finite frames only. A (finite) frame is a pair $\mathcal{T} = (T, <)$ where $T = [0, \ldots, n]$ is an initial segment of the non-negative integers and $<$ is the usual ordering over integers. The elements of T are called *time points*. A *model* is a pair $\mathcal{M} = (\mathcal{T}, V)$ where $\mathcal{T}$ is such a frame and V (called the *valuation*) is a

mapping from propositional letters in $\mathcal{L}$ to subsets of T. Intuitively, $V(p)$ defines the set of points at which an atomic proposition $p \in \mathcal{L}$ is true.

We write $\mathcal{M}, t \models \varphi$ to express that the formula φ is *true* at time point t in the model $\mathcal{M}$. This notion of truth in a model is defined inductively over the structure of formulas:

- $\mathcal{M}, t \models p$ iff $t \in V(p)$ for propositional letters $p \in \mathcal{L}$;
- $\mathcal{M}, t \models \neg\varphi$ iff $\mathcal{M}, t \not\models \varphi$;
- $\mathcal{M}, t \models \varphi \wedge \psi$ iff $\mathcal{M}, t \models \varphi$ and $\mathcal{M}, t \models \psi$;
- $\mathcal{M}, t \models \varphi$ UNTIL ψ iff there exists a $t' \in T$ with $\mathcal{M}, t' \models \psi$ and $t < t'$, and $\mathcal{M}, t'' \models \varphi$ for all $t'' \in T$ with $t < t''$ and $t'' < t'$.

Propositional connectives other than negation and conjunction can be defined in the usual manner; e.g. $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$. We also use $\top$ as a shorthand for $p \vee \neg p$ for some propositional letter p, *i.e.* $\top$ is true at any point in a model. The symbol $\bot$ is short for $\neg\top$. Further temporal operators can be defined in terms of the UNTIL-operator:

$$\bigcirc\varphi = \bot \text{ UNTIL } \varphi$$
$$\Diamond\varphi = \top \text{ UNTIL } \varphi$$
$$\Box\varphi = \neg\Diamond\neg\varphi$$

The first of these is called the next-operator: $\bigcirc\varphi$ is true at t whenever φ is true at a future point t' and there are no other points in between t and t' (as they would have to satisfy $\bot$), *i.e.* φ is true at the *next* point in time. The eventuality operator $\Diamond$ is used to express that a formula holds at *some* future time, while $\Box\varphi$ says that φ is true *always* in the future (it is not the case that there exists a future point where φ is not true). Alternatively, in particular if we are working with a fragment of PLTL that may not include the UNTIL-operator, these modalities can also be defined directly.

4 Dialogues as Models

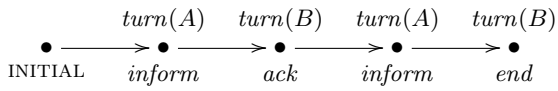
Given a model $\mathcal{M}$ and a formula φ, the *model checking* problem is the problem of deciding whether φ is true at every point in $\mathcal{M}$. In the sequel, we are going to formulate the problem of checking conformance of a dialogue to a protocol as a (variant of the) model checking problem. The extraordinary success of model checking in software engineering in recent years is largely due to the availability of very efficient algorithms, in particular for the branching-time temporal logic CTL [8]. Given that the reasoning problems faced in the context of agent communication will typically be considerably less complex than those encountered in software engineering, efficiency is not our main concern. Instead, clarity and simplicity of protocol specifications must be our main objective.

We are going to use a special class of PLTL models to represent dialogues between agents and PLTL formulas to specify protocols. For every agent A referred to in the protocol under consideration, we assume that the set $\mathcal{L}$ of propositional letters includes a special proposition $turn(A)$ and that there are no other propositions of this form in $\mathcal{L}$. Furthermore, we assume that the set of performatives

in our communication language is a subset of $\mathcal{L}$, and that $\mathcal{L}$ includes the special proposition INITIAL. We say that a model *represents* a dialogue iff it meets the following conditions:

- INITIAL is true at point 0 and at no other $t > 0$;
- exactly one proposition of the form $turn(\_)$ is true at any point $t > 0$;
- exactly one performative is true at any point $t > 0$.

Note that we do not allow for concurrent moves. The following is an example for such a model representing a dialogue (conforming to the protocol of Figure 1):



An actual dialogue determines a *partial* model: It fixes the frame as well as the valuation for INITIAL and the propositions in $\mathcal{L}$ corresponding to turn-assignments and performatives, but it does not say anything about any of the other propositional letters that we may have in our language $\mathcal{L}$ (e.g. to represent dialogue states; see Section 5). We can *complete* a given partial model by arbitrarily fixing the valuation V for the remaining propositional letters. Every possible way of completing a dialogue model in this manner givens rise to a different PLTL model, *i.e.* a dialogue typically corresponds to a whole classes of models. This is why we cannot use standard model checking (which applies to single models) to decide whether a given dialogue satisfies a formula encoding a protocol. Instead, the reasoning problem we are interested in is this:

> *Given a partial model $\mathcal{M}$ (induced by a dialogue) and a formula φ (the specification of a protocol), is there a full model $\mathcal{M}'$ completing $\mathcal{M}$ such that φ is true at every point in $\mathcal{M}'$?*

In other words, we have to decide whether the partial description of a model can be completed in such a way that model checking would succeed.

The above problem is known as the *generalised model checking* problem and has been studied by Bruns and Godefroid [7]. In fact, the problem they address is slightly more general than ours, as they do not work with a fixed frame and distinguish cases where all complete instances of the partial model validate the formula from those where there exists at least one such instance. Generalised model checking may be regarded as a combination of satisfiability checking and model checking in the usual sense. If there are no additional propositions in $\mathcal{L}$, then generalised model checking reduces to standard model checking. If we can characterise the class of all models representing a given dialogue by means of a formula ψ, then φ and ψ can be used to construct a formula that is satisfiable (has got a model) iff that dialogue conforms to the protocol given by φ.

Note that the generalised model checking problem is EXPTIME-complete for both CTL and PLTL [7], *i.e.* there would be no apparent computational advantages in using a branching-time logic.

Before we move on to show how PLTL can be used to specify protocols in Sections 5 and 6, one further technical remark is in order. While we have defined the semantics of PLTL with respect to *finite* frames, the standard model checking algorithms for this logic are designed to check that all *infinite* runs through a given Kripke structure satisfy the formula in question. This is a crucial feature of these algorithms as they rely on the translation of temporal logic formulas into Büchi automata [21, 34] and acceptance conditions for such automata are defined in terms of states that are being visited infinitely often. We note here that the problem of (generalised) model checking for finite models admitting only a single run is certainly not more difficult than (generalised) model checking for structures with infinite runs. Furthermore, to directly exploit existing algorithms, our approach could easily be adapted to a representation of dialogues as structures admitting only infinite runs. Because our main interest here lies in representing communication protocols and highlighting the potential of automated reasoning tools in this area, rather than in the design of concrete algorithms, in the remainder of the paper, we are going to continue to work with finite models.

5 Automata-based Protocols

A wide range of communication protocols studied in the multiagent systems literature can be represented using deterministic finite automata (see e.g. [12, 27–29]). As we shall see, we can represent this class of protocols using a fragment of PLTL where the only temporal operator required is the next-operator $\bigcirc$.

Consider again the protocol of Figure 1, which is an example for such an automaton-based protocol. If our language $\mathcal{L}$ includes a propositional letter of the form $state(i)$ for every state $i \in \{0, \ldots, 4\}$, then we can describe the *state transition function* of this automaton by means of the following formulas:

$$state(0) \wedge \bigcirc inform \rightarrow \bigcirc state(1)$$
$$state(1) \wedge \bigcirc ack \rightarrow \bigcirc state(2)$$
$$state(1) \wedge \bigcirc end \rightarrow \bigcirc state(3)$$
$$state(2) \wedge \bigcirc inform \rightarrow \bigcirc state(1)$$
$$state(2) \wedge \bigcirc end \rightarrow \bigcirc state(4)$$

To specify that state 0 is the (only) initial state we use the following formula:

$$\textsc{initial} \leftrightarrow state(0)$$

For automata with more than one initial state, we would use a disjunction on the righthand side of the above formula.

Next we have to specify the range of legal follow-up moves for every dialogue state. Let us ignore, for the moment, the question of turn-taking and only consider performatives. For instance, in state 1, the only legal follow-up moves would be *ack* and *end*. The seemingly most natural representation of this legality condition would be the following:

$$state(1) \rightarrow \bigcirc(ack \vee end)$$

This representation is indeed useful if we want to verify the legality of a *complete* dialogue. However, if we also want to use (generalised) model checking to establish whether an *unfinished* dialogue conforms to a protocol, we run into problems. Take a dialogue that has just begun and where the only event so far is a single *inform* move uttered by agent A, *i.e.* we are in state 1 and the dialogue should be considered legal, albeit incomplete. Then the next-operator in the above legality condition would *force* the existence of an additional time point, which is not present in the dialogue model under consideration, *i.e.* model checking would fail.

To overcome this problem, we use a *weak* variant of the next-operator. Observe that a formula of the form $\neg \bigcirc \neg \varphi$ is true at time point t iff φ is true at the successor of t or t has no successor at all. For the non-final states in the protocol of Figure 1, we now model *legality conditions* as follows:

$$\begin{aligned} &\mathit{state}(0) \rightarrow \neg \bigcirc \neg \mathit{inform} \\ &\mathit{state}(1) \rightarrow \neg \bigcirc \neg (\mathit{ack} \vee \mathit{end}) \\ &\mathit{state}(2) \rightarrow \neg \bigcirc \neg (\mathit{inform} \vee \mathit{end}) \end{aligned}$$

Next we specify that states 3 and 4 are *final* states and that a move taking us to a final state cannot have any successors:

$$\begin{aligned} &\textsc{final} \leftrightarrow \mathit{state}(3) \vee \mathit{state}(4) \\ &\textsc{final} \rightarrow \neg \bigcirc \top \end{aligned}$$

Automata-based protocols regulating the communication between pairs of agents will typically implement a strict *turn-taking policy* (although this need not be so; see [27] for an example). This is also the case for the continuous update protocol. After a dialogue has been initiated, it is agent A's turn and after that the turn changes with every move. This can be specified as follows:

$$\begin{aligned} &\textsc{initial} \rightarrow \neg \bigcirc \neg \mathit{turn}(A) \\ &\mathit{turn}(A) \rightarrow \neg \bigcirc \neg \mathit{turn}(B) \\ &\mathit{turn}(B) \rightarrow \neg \bigcirc \neg \mathit{turn}(A) \end{aligned}$$

Alternatively, these rules could have been incorporated into the specification of legality conditions pertaining to performatives given earlier. Where possible, it seems advantageous to separate the two, to allow for a modular specification.

Now let φ_{cu} stand for the conjunction of the above formulas characterising the continuous update protocol (*i.e.* the five formulas encoding the transition function, the formulas characterising initial and final states, the three formulas specifying the legality conditions for non-final states, and the formulas describing the turn-taking policy). Then a (possibly incomplete) dialogue is legal according to this protocol iff generalised model checking succeeds for φ_{cu} with respect to the partial model induced by the dialogue.

If we want this check to succeed only if the dialogue is not only legal but also complete, we can add the following formulas, which specify that any non-final state requires an additional turn:

$$\begin{aligned} &\textsc{non-final} \leftrightarrow \mathit{state}(0) \vee \mathit{state}(1) \vee \mathit{state}(2) \\ &\textsc{non-final} \rightarrow \bigcirc \top \end{aligned}$$

While our description of how to specify automata-based protocols in PLTL has been example-driven, the general methodology is clear: It involves the specification of both the state transition function (including the identification of initial, final, and non-final states) and the range of legal follow-ups for any given state.

A special class of automata-based protocols, so-called *shallow* protocols, have been identified in [12]. A shallow protocol is a protocol where the legality of a move can be determined on the sole basis of the previous move in the dialogue. Many automata-based protocols in the multiagent systems literature, including the continuous update protocol and those proposed in [27–29], are shallow and allow for an even simpler specification than the one presented here. In fact, these protocols can be specified using a language $\mathcal{L}$ including *only* the special symbol INITIAL and propositions for performatives and turn-assignment (along the lines of the rules for the turn-taking policy given earlier), *i.e.* for this class of protocols standard model checking may be used to check conformance. Where available, a shallow specification may therefore be preferred.

6 Modelling Future Obligations

For many purposes, purely automata-based protocols are not sufficient. For instance, they do not support the specification of general *future obligations* on the communicative behaviour of an agent. This is an important feature of many classes of protocols proposed in the literature. Examples are the *discourse obligations* of Traum and Allen [33], the *commitments* in the work of Singh [31] and Colombetti [9], or the *social expectations* of Alberti et al. [1].

We should stress that we use the term *obligation* in rather generic a manner; in particular, we are not concerned with the fine distinctions between, say, obligations and commitments discussed in the literature [9, 30].

In the context of an auction protocol, for example, we may say that, by opening an auction, an auctioneer acquires the obligation to close that auction again at some later stage. Suppose these actions can be performed by making a dialogue move with the performatives *open-auction* and *end-auction*, respectively. Again, to simplify presentation, we abstract from the issue of turn-taking and only write rules pertaining to performatives. The most straightforward representation of this protocol rule would be the following:

$$\textit{open-auction} \rightarrow \Diamond \textit{end-auction}$$

However, in analogy to the problematic aspects of using the next-operator to specify legal follow-ups in the context of automata-based protocols, the above rule forces the existence of future turns in a dialogue once *open-auction* has been performed. If we were to check an incomplete dialogue against this specification before the auction has been closed, model checking would fail and the dialogue would have to be classified as illegal. To be able to distinguish between complete dialogues where the non-fulfilment of an obligation constitutes a violation of the protocol and incomplete dialogues where this may still be acceptable, we have to move to a slightly more sophisticated specification.

To this end, we first define a weak version of the UNTIL-operator, which is sometimes called the UNLESS-operator:

$$\varphi \text{ UNLESS } \psi = (\varphi \text{ UNTIL } \psi) \vee \Box\varphi$$

That is, the formula φ UNLESS ψ is true at point t iff φ holds from t onwards (excluding t itself) either until a point where ψ is true or until the last point in the model.

We now use the following formula to specify that opening an auction invokes the obligation to end that auction at some later point in time:

$$\textit{open-auction} \rightarrow \text{PENDING} \ \wedge \ (\text{PENDING UNLESS } \textit{end-auction})$$

The new propositional letter PENDING is used to mark time points at which there are still obligations that have not yet been fulfilled. A model representing a dialogue where *open-auction* has been uttered, but *end-auction* has not, will satisfy this protocol rule. However, in such a model, PENDING will be true at the very last time point. If we want to check whether a dialogue does not only not violate any rules but also fulfils all obligations, we can run generalised model checking with a specification including the following additional formula:

$$\text{PENDING} \rightarrow \bigcirc\top$$

No finite model satisfying this formula can make PENDING true at the last time point. That is, unless *end-auction* has been uttered, generalised model checking will now fail.

In a slight variation of our example, we may require our agent to end the auction not just at some point in the future, but by a certain deadline. Reference to concrete time points ("by number") is something that is typically not possible (nor intended) in temporal logic. However, if we can model the invocation of the deadline by means of a proposition *deadline* (which could be, say, the logical consequence of another agent's dialogue moves), then we can add the following formula to our specification to express that *end-auction* has to be uttered before *deadline* becomes true:

$$\textit{open-auction} \rightarrow (\neg \textit{deadline} \text{ UNLESS } \textit{end-auction})$$

The examples in this section suggest that PLTL is an appropriate language for specifying dialogue obligations. Due to Kamp's seminal result on the expressive completeness of PLTL over Dedekind-complete flows of time (which include our finite dialogue frames), we know that we can express *any* combination of temporal constraints over obligations expressible in the appropriate first-order theory also in PLTL [16, 25].

Of course, protocol rules that constrain the *content* item in a dialogue move (e.g. "the price specified in a bid must be higher than any previous offer") cannot be represented in PLTL, nor in any other general-purpose logic. Arguably, while (temporal) logic is a suitable tool for modelling conversational conventions, reasoning about application-specific content requires domain-specific reasoners (even in simple cases such as the comparison of alternative price offers).

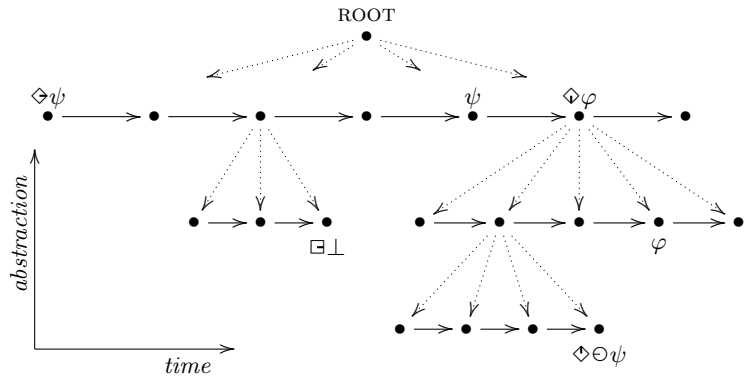


Fig. 2. An ordered tree model

7 Nested Protocols

In practice, a multiagent system may specify a whole range of different interaction protocols, and agents may use a combination of several of these during a communicative interaction [28, 36]. For instance, there may be different protocols for different types of auctions available, as well as a meta-protocol to jointly decide which of these auction protocols to use in a given situation. Such *nesting* of protocols could also be recursive.

We propose to use OTL [10, 11], a *modal logic of ordered trees* (see also [4, 5]), to specify nested protocols. This is an extended temporal logic based on frames that are ordered trees, *i.e.* trees where the children of each node form a linear order. In the context of modelling dialogues, again, we may assume that such trees are finite. OTL is the modal logic over frames that are ordered trees. The logic includes modal operators for all four directions in an ordered tree. The formula $\ominus\varphi$, for instance, expresses that φ is true at the immediate righthand sibling of the current node, while $\boxdot\psi$ forces ψ to be true at all of its children.

We briefly summarise the syntax and semantics of OTL; details may be found in [10]. The set of *formulas* of OTL is the smallest set extending the language of classical propositional logic such that. whenever φ is a formula, so are $\odot\varphi$, $\Diamond\!\!\!\!\cdot\,\varphi$, $\ominus\varphi$, $\Diamond\!\!\!\!\cdot\,\varphi$, $\ominus\varphi$, $\Diamond\!\!\!\!\cdot\,\varphi$, $\Diamond\!\!\!\!\cdot\,\varphi$ and $\Diamond\!\!\!\!\cdot^{+}\varphi$ (we omit the discussion of *until*-style operators from this short introduction [4, 10]). Formulas are evaluated over ordered trees. An ordered tree $\mathcal{T}$ defines the relations of being a *parent*, *child*, *ancestor*, *descendant*, *lefthand* and *righthand sibling* over a set of nodes T. The first sibling to the left of a node is also called that node's *lefthand neighbour* (and *righthand neighbours* are defined analogously). An *ordered tree model* is a pair $\mathcal{M} = (\mathcal{T}, V)$ where $\mathcal{T}$ is such an ordered tree an V is a valuation function from propositional letters to subsets of T. The truth conditions for atomic formulas and the propositional connectives are defined as for PLTL. Furthermore:

– $\mathcal{M}, t \models \ominus\varphi$ iff t is not the root of $\mathcal{T}$ and $\mathcal{M}, t' \models \varphi$ holds for t's parent t';
– $\mathcal{M}, t \models \Diamond\!\!\!\!\!-\varphi$ iff t has got an ancestor t' such that $\mathcal{M}, t' \models \varphi$;
– $\mathcal{M}, t \models \ominus\varphi$ iff t has got a righthand neighbour t' such that $\mathcal{M}, t' \models \varphi$;
– $\mathcal{M}, t \models \Diamond\!\!\!\!\!\rightarrow\varphi$ iff t has got a righthand sibling t' such that $\mathcal{M}, t' \models \varphi$;
– $\mathcal{M}, t \models \Diamond\!\!\!\!\!-\varphi$ iff t has got a child t' such that $\mathcal{M}, t' \models \varphi$;
– $\mathcal{M}, t \models \Diamond\!\!\!\!\!-^{+}\varphi$ iff t has got a descendant t' such that $\mathcal{M}, t' \models \varphi$.

The truth conditions for $\Diamond\!\!\!\!\!\rightarrow$ and $\ominus$ are similar. Box-operators are defined in the usual manner: $\boxminus\varphi = \neg\Diamond\!\!\!\!\!-\neg\varphi$, etc. The semantics explains our choice of a slightly different notation for the downward modalities: because there is (usually) no *unique* next node when moving down in a tree, we do not use a next-operator to refer to children. Figure 2 shows an example for an ordered tree model.

This logic can be given a *temporal interpretation*. Time is understood to run from left to right, along the order declared over the children of a node (*i.e. not* from top to bottom as in branching-time logics such as CTL), while the child relation provides a means of "zooming" into the events associated with a node. In the context of dialogues and nested protocols, the righthand sibling relation is used to model the passing of time with respect to a single protocol, while the child relation is used to model the relationship between a dialogue state and the subprotocol being initiated from that state.

Our example for a nested protocol is inspired by work in natural language dialogue modelling [13, 17]. When a question is asked, besides answering that question, another reasonable follow-up move would be to pose a clarification question related to the first question. This latter question would then have to be answered before the original one. This protocol rule may, in principle, be applied recursively, *i.e.* we could have a whole sequence of clarification questions followed by the corresponding answers in reverse order. In addition, we may also ask several clarification questions pertaining to the same question (at the same level). The corresponding protocol is shown in Figure 3. The edge labelled by $\textsc{clar}(B, A)$ represents a "meta-move": this is not a dialogue move uttered by one of the agents involved, but stands for a whole subdialogue following the rules of the clarification protocol with B (rather than A) being the agent asking the initial question. That is, the clarification protocol of Figure 3 does *not* belong to the class of automata-based protocols discussed in Section 5. Indeed, this kind of protocol cannot be specified by a simple finite state automaton. Instead we would require a pushdown automaton [26]. The stack of such a pushdown automaton would be used to store questions and every answer would cause the topmost question to be popped again [13].

Suppose INITIAL, FINAL, and NON-FINAL states have been specified as in Section 5 (using $\ominus$ in place of $\bigcirc$). If we treat CLAR in the same way as we would treat a simple performative, then the transition function for the clarification protocol can be specified as follows:

$$\begin{aligned} &\mathit{state}(0) \wedge \ominus \mathit{ask} \rightarrow \ominus \mathit{state}(1) \\ &\mathit{state}(1) \wedge \ominus \textsc{clar} \rightarrow \ominus \mathit{state}(1) \\ &\mathit{state}(1) \wedge \ominus \mathit{answer} \rightarrow \ominus \mathit{state}(2) \end{aligned}$$

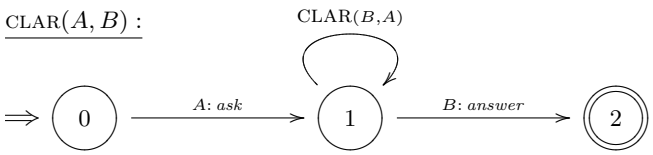


Fig. 3. A clarification protocol

Abstracting from turn-taking issues, the legality conditions for this protocol are given by the following formulas:

$$state(0) \rightarrow \neg \ominus \neg ask$$
$$state(1) \rightarrow \neg \ominus \neg (\textsc{clar} \vee answer)$$

The next formula says that a node corresponding to a final state in a subdialogue cannot have any righthand siblings:

$$\textsc{final} \rightarrow \neg \ominus \top$$

That CLAR requires a subdialogue to take place can be specified by a formula that says that every node satisfying CLAR has to have a child satisfying INITIAL:

$$\textsc{clar} \rightarrow \diamondsuit \textsc{initial}$$

The next formula specifies that a subdialogue must be completed before the dialogue at the next higher level may continue. This is expressed by postulating that, if a node has got a righthand sibling, then its rightmost child (if any) cannot satisfy NON-FINAL:

$$\ominus \top \rightarrow \boxdot (\textsc{non-final} \rightarrow \ominus \top)$$

To characterise dialogues that have been completed in their entirety, we may again add the following rule:

$$\textsc{non-final} \rightarrow \ominus \top$$

We hope that this very simple example gives some indication of the options available to us when specifying nested protocols using OTL. Our example has been special in the sense that it only uses a single protocol that can be nested arbitrarily. In general, there may be several different protocols, each associated with its own propositions to identify initial, final, and non-final states. Observe that, for OTL, deriving the partial (ordered tree) model induced by an observed dialogue is not as straightforward as for PLTL. However, if the moves used to initiate and terminate subdialogues following a particular protocol clearly identify that protocol (which seems a reasonable assumption), then constructing an ordered tree from a sequence of utterances is not difficult.

For OTL, to date, no model checking algorithms (or algorithms for generalised model checking) have been developed. However, it seems likely that such algorithms could be designed by adapting well-known algorithms for other temporal logics. And even without the availability of tools for model checking, we believe that the specification of nested protocols in OTL can be useful to give a precise semantics to the intuitive "operation of nesting".

8 Conclusion

In this paper, we have argued that temporal logic can be used to specify convention-based agent communication protocols in a simple and elegant manner. In particular, we have seen how to use *propositional linear temporal logic* to specify both very simple *automata-based protocols* and protocols involving *dialogue obligations*. Of course, using this logic to express the kinds of properties we have considered in our examples is not new, but the application of this technique to the specification of conversational conventions is both novel and, we believe, very promising. We have then outlined how *nested protocols* can be specified using the *ordered tree logic* OTL, which is an extension of propositional linear temporal logic, but also permits reasoning about different levels of abstraction within a single model.

We have also identified *generalised model checking* as a tool for checking protocol conformance at runtime. For simple protocol representation formalisms (such as finite automata), this is not a difficult problem and to resort to sophisticated tools such a model checking may seem inappropriate. However, for richer formalisms, in particular those that allow for the definition of complex dialogue obligations, the problem is certainly not trivial (witness the work of Alberti et al. [1], who develop a complex abductive proof procedure to address conformance checking). Computational issues aside, being able to define the conformance problem in clear logical terms already constitutes an important advantage in its own right.

Our aim for this paper has been to promote the use of simple temporal logics in the context of agent communication. Most of our presentation has been based on examples, but we hope that the generality of the approach shines through. As argued already at the end of Section 6, linear temporal logic is very expressive and can specify a rich class of protocols. Our concrete examples merely highlight some of the most important features of typical protocols.

The idea of using temporal logic for the representation of convention-based agent interaction protocols is not new [32, 35]. The two cited works both use a form of the branching-time temporal logic CTL to give semantics to the notion of *social commitment*, but they do not attempt to exploit existing automated reasoning tools developed for these logics. The logic of Verdicchio and Colombetti [35] also incorporates some, albeit very restricted, first-order features. In our view, this is unfortunate as it trades in much of what is attractive about using temporal logics (decidability, low complexity, simple semantics).

Although there has been a growing interest in model checking for multiagent systems in recent years (examples include [3, 6, 24]), only little work has specifically addressed issues of communication. An exception is the work of Huget and Wooldridge [22], which studies model checking as a tool for verifying conformance to the semantics of an agent communication language in a mentalistic framework. There has also been a certain amount of work on *deductive* approaches to verification in multiagent systems [14], but again without special focus on communication protocols or conversational conventions.

In our future work, we hope to cover a wider range of protocol features and show how they may be specified using a suitable temporal logic. For instance, it would be interesting to explore the use of past-time operators to specify protocol rules relating to the content of a *commitment store* (e.g. only challenge arguments that have previously been asserted), as used in the context of argumentation-based communication models [2, 13, 20].

References

1. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interactions using social integrity constraints. In *Workshop on Logic and Communication in Multi-Agent Systems*, 2003.
2. L. Amgoud, N. Maudet, and S. Parsons. Modelling dialogues using argumentation. In *4th International Conference on MultiAgent Systems*. IEEE, 2000.
3. M. Benerecetti, F. Giunchiglia, and L. Serafini. Model checking multiagent systems. *Journal of Logic and Computation*, 8(3):401–423, 1998.
4. P. Blackburn, B. Gaiffe, and M. Marx. Variable-free reasoning on finite trees. In *Mathematics of Language 8*, 2003.
5. P. Blackburn, W. Meyer-Viol, and M. de Rijke. A proof system for finite trees. In *Computer Science Logic*. Springer-Verlag, 1996.
6. R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In *2nd International Conference on Autonomous Agents and Multiagent Systems*. ACM Press, 2003.
7. G. Bruns and P. Godefroid. Generalized model checking: Reasoning about partial state spaces. In *11th International Conference on Concurrency Theory*. Springer-Verlag, 2000.
8. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
9. M. Colombetti. A commitment-based approach to agent speech acts and conversations. In *Workshop on Agent Languages and Conversation Policies*, 2000.
10. U. Endriss. *Modal Logics of Ordered Trees*. PhD thesis, King's College London, Department of Computer Science, 2003.
11. U. Endriss and D. Gabbay. Halfway between points and intervals: A temporal logic based on ordered trees. In *ESSLLI Workshop on Interval Temporal Logics and Duration Calculi*, 2003.
12. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *18th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 2003.
13. R. Fernández and U. Endriss. Towards a hierarchy of abstract models for dialogue protocols. In *Proceedings of the 5th International Tbilisi Symposium on Language, Logic and Computation*. ILLC, 2003.

14. M. Fisher. Temporal development methods for agent-based systems. *Journal of Autonomous Agents and Multi-agent Systems*, 10:41–66, 2005.
15. Foundation for Intelligent Physical Agents (FIPA). *Communicative Act Library Specification*, 2002.
16. D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*, volume 1. Oxford University Press, 1994.
17. J. Ginzburg. Interrogatives: Questions, facts, and dialogue. In *Handbook of Contemporary Semantic Theory*. Blackwell, 1996.
18. R. Goldblatt. *Logics of Time and Computation*. CSLI, 2nd edition, 1992.
19. F. Guerin and J. Pitt. Guaranteeing properties for *e*-commerce systems. In *Agent-Mediated Electronic Commerce IV*. Springer-Verlag, 2002.
20. C. L. Hamblin. *Fallacies*. Methuen, London, 1970.
21. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, pages 279–295, 1997.
22. M.-P. Huget and M. Wooldridge. Model checking for ACL compliance verification. In *Advances in Agent Communication*. Springer-Verlag, 2004.
23. A. J. I. Jones and X. Parent. Conventional signalling acts and conversation. In *Advances in Agent Communication*. Springer-Verlag, 2004.
24. M. Kacprzak, A. Lomuscio, and W. Penczek. Verification of multiagent systems via unbounded model checking. In *3rd International Conference on Autonomous Agents and Multiagent Systems*. ACM Press, 2004.
25. J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California at Los Angeles, Department of Philosophy, 1968.
26. H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall International, 2nd edition, 1998.
27. S. Parsons, N. Jennings, and C. Sierra. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.
28. J. Pitt and A. Mamdani. Communication protocols in multi-agent systems. In *Workshop on Specifying and Implementing Conversation Policies*, 1999.
29. J. Pitt and A. Mamdani. A protocol-based semantics for an agent communication language. In *16th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1999.
30. M. J. Sergot. A computational theory of normative positions. *ACM Transactions on Computational Logic*, 2(4):581–622, 2001.
31. M. P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.
32. M. P. Singh. A social semantics for agent communication languages. In *Issues in Agent Communication*. Springer-Verlag, 2000.
33. D. R. Traum and J. F. Allen. Discourse obligations in dialogue processing. In *32nd Annual Meeting of the Association for Computational Linguistics*, 1994.
34. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *1st Symposium on Logic in Computer Science*. IEEE, 1986.
35. M. Verdicchio and M. Colombetti. A logical model of social commitment for agent communication. In *2nd International Conference on Autonomous Agents and Multiagent Systems*. ACM Press, 2003.
36. B. Vitteau and M.-P. Huget. Modularity in interaction protocols. In *Advances in Agent Communication*. Springer-Verlag, 2004.
37. M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2002.