Symbolic Model Checking Commitment Protocols Using Reduction

Mohamed El-Menshawy[1], Jamal Bentahar[1], and Rachida Dssouli[2]

[1] Concordia University, Faculty of Engineering and Computer Science, Canada
[2] Concordia University, Canada and UAE University, Faculty of Inf. Tech., UAE
m_elme@encs.concordia.ca, bentahar@ciise.concordia.ca,
dssouli@ciise.concordia.ca

Abstract. Using model checking to verify that interaction protocols have given properties is widely recognized as an important issue in multi-agent systems where autonomous and heterogeneous agents need to successfully regulate and coordinate their interactions. In this paper, we investigate the use of symbolic model checkers to verify the compliance of a special kind of interaction protocols called commitment protocols with some properties such as liveness and safety. These properties are expressed as formulae in a new temporal logic, called CTLC, which extends the temporal logic CTL with modality for social commitments. Our approach shows that the problem of model checking CTLC can be reduced to the problem of model checking either CTLK or ARCTL, which are extensions of CTL. We finally present an implementation and report on the experimental results of verifying the Contract Net Protocol modeled in terms of commitments and associated actions using the symbolic model checkers MCMAS and extended NuSMV.

Keywords: Multi-Agent Systems, Commitment Protocols, Symbolic Model Checking, Protocol Properties.

1 Introduction

Over the last two decades, the researchers on Multi-Agent Systems (MASs) have been focused both on defining a clear and standard semantics for Agent Communication Languages (ACLs), such as FIPA-ACL[1], and developing multi-agent interaction protocols. The developers of FIPA-ACL have addressed the challenge of incorporating ACL and protocols by proposing a set of multi-agent interaction protocols, called FIPA-ACL protocols[2]. These protocols can be viewed as specific ACLs designed for particular purposes such as *Request Interaction Protocol (RIP)*, *English Auction Interaction Protocol (EAIP)* and *Contract Net Protocol (CNP)*. In

[1] This term stands for the Foundation for Intelligent Physical Agents' Agent Communication Language–see for examples, FIPA-ACL specifications (1997, 1999, 2001, 2002), http://www.fipa.org/repository/aclspecs.php3

[2] See for examples, FIPA-ACL Interaction Protocols (2001, 2002), http://www.fipa.org/repository/ips.php3

A. Omicini, S. Sardina, and W. Vasconcelos (Eds.): DALT 2010, LNAI 6619, pp. 185–203, 2011.

particular, CNP is designed from online business point of view to reach agreements among interacting agents. FIPA-ACL protocols have succeed in specifying the rules governing interactions and coordinating dialogues among agents by: 1) restricting the range of allowed follow-up communicative acts at any stage during a dialogue; and 2) describing the sequence of messages that FIPA compliant agents can exchange for particular applications. However, these protocols are quite rigid to be used by autonomous agents (that do what is best for themselves) as they are specified so that agents must execute them without possibility of handling exceptions that appear at run time, which restricts the protocols' flexibility.

Recently, social approaches have been proposed to overcome FIPA-ACL protocols' shortcomings. In particular, social approaches advocate declarative representations of protocols and give semantics to protocol messages in terms of social concepts. Bentahar et al. [2] have proposed a framework capable of specifying effective multi-agent interaction protocols using a combination of argumentation theory and social commitments. Fornara and Colombetti [12] have based the semantics of agent communication protocols on social commitments such that the meanings of exchanged messages are denoted by social commitments and their associated actions. Yolum and Singh [29] have developed an approach to flexibly specify multi-agent interaction protocols wherein protocols capture the dynamic behaviors of the agents in terms of creation and manipulation of commitments to one another. All these protocols have the characteristic of being commitment-based and are called commitment protocols. Furthermore, Chopra, Yolum and Singh have developed a formalism to represent and reason about commitment protocols called *commitment machines* based on either *event calculus* or *non-monotonic theory* of actions in terms of causal logic [28,6]. This formalism can represent flexible protocols that enable agents to exercise their autonomy by dealing with exceptions and making choices. In the same line of research, Singh [24] has generalized the formalism of commitment machines to include natural non-terminal protocols (or protocols that have cycles) analogous to those in real-life business applications.

In addition to providing flexibility during run time, these approaches make it possible to provide a meaningful basis for compliance of agents with a given protocol. This is because commitments can be stored publicly (or observed by all participating agents) and agents that do not satisfy their commitments at the end of the protocol can be identified as non-compliant [25,7,26]. In order for these approaches to make use of all these advantages, they should integrate rigorous design and automatic verification of interaction protocols within the same framework. For instance, Venkatraman et al. [25] have presented an approach for locally testing whether or not the behavior of an agent in open systems complies with a given commitment protocol specified in Computational Tree Logic (CTL). Cheng [5] and Desai et al. [10] have used OWL-P to specify commitment protocols and their compositions. To verify their protocols against some properties expressed in Linear Temporal Logic (LTL), they translate them into PROMELA code, which is the input language of the automata-based model checker SPIN. Yolum [27] has defined three "generic properties" taken from distributed systems

that can be incorporated in a design tool to "semi-automate" the specification of commitment protocols at design time.

Motivation. In this paper, we aim to introduce CTLC, a CTL-like logic for social commitments. We present a fully-automatic verification technique of commitment protocols specified on the basis of this logic using symbolic model checking. This is done by introducing a mechanism to reduce the problem of model checking CTLC to the problem of model checking either CTLK [21], to directly use the MCMAS symbolic model checker [17], or ARCTL [20] to use the extended version of the NuSMV symbolic model checker introduced in [16]. The present paper inspires by the methodology introduced in [16] to perform the reduction. Finally, we present experimental results for the verification of the Contract Net Protocol, taken from e-business domain as a motivating example and specified in the proposed logic, against some desirable properties using MCMAS and the extended version of NuSMV.

Overview of Paper. The remainder of this paper is organized as follows. We begin in Section 2 by presenting the definition of social commitments and briefly summarizing the formalism of the interpreted systems used as the model of our CTLC logic. We then discuss generally the problem of model checking using MCMAS and NuSMV. In Section 3, we present CTLK and ARCTL and how the problem of model checking CTLC can be reduced to the problem of model checking either CTLK or ARCTL. Thereafter, we proceed to introduce commitment protocols and their translation along with expressing some properties in Section 4. The experimental results of verifying the Contract Net Protocol using MCMAS and the extended version of NuSMV is discussed in Section 5. In Section 6, we discuss relevant literature. We conclude the paper in Section 7.

2 Preliminaries

2.1 Commitments and Associated Actions

Social commitments have been recently gained attentions in MASs community. This is because they are formal and concise methods for describing how autonomous and heterogeneous agents communicate with one another. In particular, a social commitment is an engagement in the form of business contract between two agents: a creditor who commits to a course of action and a debtor on behalf of whom the action is done. In this paper, we distinguish between two types of commitments: unconditional commitment and conditional commitment that we need to represent commitment protocols.

Notation 1. *Unconditional commitments are denoted by* $\mathsf{C}(i, j, \varphi)$*, where* i *is the debtor,* j *is the creditor and* φ *is a well-formed formula (wff) in the proposed CTLC logic representing the commitment content.* $\mathsf{C}(i, j, \varphi)$ *means* i *socially (i.e., publicly) commits to* j *that* φ *holds.*

Notation 2. *Conditional commitments are denoted by* $\psi \rightarrow \mathsf{C}(i, j, \varphi)$*, where "*$\rightarrow$*" is the logical implication,* i*,* j *and* φ *have the above meanings and* ψ *is a wff in the proposed CTLC logic representing the commitment condition.*

We will use $\mathtt{CC}(i, j, \psi, \varphi)$ as an abbreviation of $\psi \rightarrow \mathtt{C}(i, j, \varphi)$. In order to manipulate social commitments during the progress of protocols, we introduce a set of associated actions (or operations), called commitment actions. These actions are used to capture dynamic behavior of participating agents. Defined in [23], these actions can be classified into two party actions and three party actions. The former ones need only two agents to be performed: $Create$, $Withdraw$, $Fulfill$, $Violate$ and $Release$. The latter ones need an intermediate agent to be completed: $Delegate$. In the following, we present the declarative representation of these actions where i, j and k denote agent names.

- $Create(i, j, \mathtt{C}(i, j, \varphi))$ to establish a new commitment.
- $Withdraw(i, j, \mathtt{C}(i, j, \varphi))$ to cancel an existing commitment.
- $Fulfill(i, j, \mathtt{C}(i, j, \varphi))$ to satisfy the commitment content.
- $Violate(i, j, \mathtt{C}(i, j, \varphi))$ to reflect there is no way to satisfy the commitment content.
- $Release(j, i, \mathtt{C}(i, j, \varphi))$ to free a debtor from carrying out his commitment.
- $Delegate(i, k, \mathtt{C}(i, j, \varphi))$ to delegate an existing commitment to another debtor to satisfy it on his behalf.

2.2 Interpreted Systems and CTLC Logic

An interpreted system as introduced by Fagin et al. [11] is a formalism that models the temporal evolution of a system of agents to reason about knowledge and temporal properties. We start with assuming that a MAS is composed of n agents $\mathcal{A} = \{1, \ldots, n\}$. Each agent i is characterized by a set of local states L_i and a set of local actions Act_i. In this paper, these actions include the commitment actions and a special action ϵ_i denoting the "null" action for agent i. Thus, when an agent performs the null action, the local state of this agent remains the same. Moreover, for each agent $i \in \mathcal{A}$, I_i defines an initial state and a local protocol $\mathcal{P}_i : L_i \rightarrow 2^{Act_i}$, which is a function that maps the current state of the agent i with the set of enabled actions for that state. The agents act within an "environment" (e), which can be also modeled with a set of local states L_e, a set of local actions Act_e and a local protocol $\mathcal{P}_e$. This can be seen as a special agent that can capture any information that may not pertain to a specific agent.

Definition 1 ([11]). *A set G of global states in a MAS is: $G \subseteq L_i \times \ldots \times L_n \times L_e$, where a state $g = (l_1, \ldots, l_n, l_e) \in G$ can be seen as a "snapshot"of all agents in the MAS at a given time and $l_i(g)$ represents the local state of agent i in the global state g.*

The evolution function that determines the transitions for an individual agent between its local states is defined as follows: $t_i : L_i \times L_e \times ACT \rightarrow L_i$, where $ACT = Act_1 \times \ldots \times Act_n \times Act_e$ and each component $a \in ACT$ is a "joint action", which is a tuple of actions (one for each agent). The global evolution function $t : G \times ACT \rightarrow G$ is defined as follows: $t(g, act_1, \ldots, act_n, act_e) = g'$ iff there exists $a \in ACT$ such that (i) for each agent i that is able to perform

a, we have $t_i(l_i, l_e, a) = l_i'$; and (ii) for each agent j that is unable to perform a, we have $t_j(l_j, l_e, \epsilon_j) = l_j$. Notice that we use a special class of interpreted systems in which at each moment only one agent can perform an action in a global evolution function and I denotes a set of initial states. Finally, given a set Φ_p of atomic propositions and the valuation function V for those propositions $V : G \rightarrow 2^{\Phi_p}$, an interpreted system is a tuple:

$$\mathcal{IS} = \Big\langle (L_i, Act_i, \mathcal{P}_i, t_i)_{i \in \mathcal{A}}, (L_e, Act_e, \mathcal{P}_e, t_e), I, V \Big\rangle$$

Computation tree logic of social commitments CTLC is an extension of CTL [9,11] with the commitment modality $\mathtt{C}(i, j, \varphi)$. In particular, the syntax of CTLC is given by the following BNF grammar, where $p \in \Phi_p$ is an atomic proposition:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{EX}\varphi \mid \mathtt{EG}\varphi \mid \mathtt{E}(\varphi \mathtt{U} \varphi) \mid \mathtt{C}(i, j, \varphi)$$

where the CTLC temporal modalities have the standard meaning as in CTL—for example, $\mathtt{EX}\varphi$ means that "there is a path where φ holds at the next state in the path". $\mathtt{C}(i, j, \varphi)$ is read as "agent i commits towards agent j to bring about φ". Other derived operators are defined in a standard way, see for example [9,11]. In order to interpret CTLC formulae, a Kripke model $M = (W, I, R_t, R_{sc}, V)$ is associated to a given interpreted system $\mathcal{IS}$ as follows:

- W is the set G of global states,
- $I \subseteq W$ is the set of initial states, which are defined in $\mathcal{IS}$,
- the temporal transition relation $R_t \subseteq W \times W$ is defined using the global evolution function t,
- the relation $R_{sc} : W \times \mathcal{A} \times \mathcal{A} \rightarrow 2^W$ is the social accessibility relation for social commitments. It is defined by $w' \in R_{sc}(w, i, j)$ iff $\exists \overline{w} : l_i(w) = l_i(\overline{w})$ and $l_j(\overline{w}) = l_j(w')$,
- V is the valuation function as defined in $\mathcal{IS}$.

Excluding the commitment modality, the semantics of CTLC formulae is defined in the model M as usual (semantics of CTL), see for example [9,11]. The notation $M, \langle w \rangle \models \varphi$ means the model M satisfies φ at a state w where $\models$ is the standard satisfaction relation. The commitment modality $\mathtt{C}(i, j, \varphi)$ is satisfied in the model M at a state w iff the content φ is true in every accessible state from this state using $R_{sc}(w, i, j)$. Formally:

$M, \langle w \rangle \models \mathtt{C}(i, j, \varphi)$ iff for all $w' \in W$, if $w' \in R_{sc}(w, i, j)$ then $M, \langle w' \rangle \models \varphi$

2.3 Model Checking Using MCMAS and NuSMV

Model checking is a method of formal verification used to verify if a system satisfies given properties. In a nutshell, the problem of model checking is: given a Kripke model M and property φ (expressed as a wff), does the model satisfy that property? If an error is located (i.e., $M \not\models \varphi$), the process will return a "counter-example" showing the steps in which the error state was reached. Otherwise,

it will return true (i.e., $M \models \varphi$). Recently, model checking has been used to verify MASs [17]. Verifying these systems is becoming more and more necessary because they are increasingly used in several applications such as web-based applications [25], business processes [5,10] and artificial institutions [26].

This paper focuses both on the symbolic model checkers MCMAS [17] and the extended version of NuSMV [16], which are built on Ordered Binary Decision Diagrams (OBDDs) that alleviate to overcome the "state-explosion" problem. In particular, MCMAS is a tool used to solve the problem of model checking MASs. MCMAS also has the following features: 1) it can check a variety of properties specified as CTL formulae, epistemic, and cooperation modalities; 2) it supports variables of the following types: Boolean, enumeration and bounded integer where arithmetic operations can be performed on bounded integers; 3) it supports counter-example/witness generation for efficient display of traces falsifying/satisfying properties; and 4) it supports fairness constraints, which are useful in eliminating bad or unwanted agents' behaviors. MCMAS uses Interpreted System Programming Language (ISPL) as an input language. A system of agents is encoded in ISPL using the interpreted system components. ISPL allows user to define atomic propositions over global states of the system. The logic formulae to be checked by MCMAS are defined over these atomic propositions.

On the other hand, the NuSMV symbolic model checker [8] is written in ANSI C. It is a reimplementation and extension of SMV, the first model checker based on OBDDs. NuSMV is able to process files written in an extension of the SMV language. In this language, it is possible to describe finite state machines by means of declaration and instantiation mechanisms and processes and to express a set of requirements in CTL and LTL. In addition to the above features, NuSMV has the same features of MCMAS as MCMAS is technically an extended version of NuSMV. NuSMV can also check Real-Time CTL specifications, which specifies discrete timing constraints. However it does not model interpreted systems as it is not specially designed for MASs but can overcome this limit by indirectly checking interpreted system properties, which are encoded into its variables.

3 Model Checking CTLC

In this section, we briefly review CTLK (a logic of time and knowledge). We then show how the problem of model checking CTLC can be reduced to the problem of model checking either CTLK or ARCTL.

3.1 CTLK Logic

CTLK [21] is an epistemic logic on branching time; it allows for the expression of properties that contain a notion of knowledge. In particular, given a set of atomic propositions Φ_p, the syntax of CTLK is given by BNF grammar as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{EX}\varphi \mid \mathtt{EG}\varphi \mid \mathtt{E}(\varphi \mathtt{U} \varphi) \mid \mathtt{K}_i\varphi$$

where the epistemic modality $\mathtt{K}_i\varphi$ is used to represent "knows" that is agent i knowing φ. As in CTL, other temporal operators can be defined in a standard

way, see for example [9,11]. To define the semantic of CTLK formulae, a Kripke model of the form $M = (S, S_0, T, \sim_i, \ldots, \sim_n, V)$ is associated to a given interpreted system $\mathcal{IS}$, where: S is a set of global states; $S_0 \subseteq S$ is a set of initial global states; $T \subseteq S \times S$ is a transition relation; $\sim_i \subseteq S \times S$ are the epistemic relations defined for all $i \in \mathcal{A}$ where $s \sim_i s'$ iff $l_i(s) = l_i(s')$; and V is the valuation function as defined in $\mathcal{IS}$.

Intuitively, the epistemic relation $s \sim_i s'$ means that the local state of the agent i in the current global state s is indistinguishable from the local state of this agent in the accessible state s'. The semantics of $\mathtt{K}_i\varphi$ is defined as follows:

$M, \langle s \rangle \models \mathtt{K}_i\varphi$ iff for all $s' \in S$ if $s \sim_i s'$ then $M, \langle s' \rangle \models \varphi$

Hereafter, we use $\widehat{\mathtt{K}}_i\varphi$ as an abbreviation of $\neg\mathtt{K}_i\neg\varphi$. Its semantics is as follows:

$M, \langle s \rangle \models \widehat{\mathtt{K}}_i\varphi$ iff for some $s' \in S$ if $s \sim_i s'$ then $M, \langle s' \rangle \models \varphi$

3.2 Reducing CTLC to CTLK

In this section, we show how the problem of model checking CTLC (see Sect.2.2) can be reduced to the problem of model checking CTLK. This reduction enables us to directly use MCMAS. The problem is as follows: given a CTLC model M_{sc} and a CTLC formula φ_{sc}, we have to define a CTLK model $M = \mathscr{F}(M_{sc})$ and a CTLK formula $\mathscr{F}(\varphi_{sc})$ such that $M_{sc} \models \varphi_{sc}$ iff $\mathscr{F}(M_{sc}) \models \mathscr{F}(\varphi_{sc})$. Let $\mathcal{A} = \{1, \ldots, n\}$ be a set of agents, and $M_{sc} = (W, I, R_t, R_{sc}, V)$ be a model for CTLC associated to the interpreted system $\mathcal{IS} = \Big\langle (L_i, Act_i, \mathcal{P}_i, t_i)_{i\in\mathcal{A}}, (L_e, Act_e, \mathcal{P}_e, t_e), I, V \Big\rangle$. The model $\mathscr{F}(M_{sc})$ is a CTLK model $M = (S, S_0, T, \{\sim_i\}_{i\in\mathcal{A}}, V)$ defined as follows:

- $S = W \cup \overline{S}$ where $\overline{S}$ is constructed as follows: for all states w and w' such that $w' \in R_{sc}(w, i, j)$ add a state $\overline{s}$ in $\overline{S}$ such that $V(\overline{s}) = V(w')$ and $l_i(\overline{s}) = l_j(w')$.
- $S_0 = I$.
- the transition relation $T = R_t \cup \overline{R}_t$ where $\overline{R}_t$ is constructed as follows: for all states w and w' such that $w' \in R_{sc}(w, i, j)$ add a transition in $\overline{R}_t$ between s ($s \in S$ and $s = w$) and the added $\overline{s}$.
- the epistemic relations $\{\sim_i\}_{i\in\mathcal{A}}$ are obtained as follows: for all w and w' such that $w' \in R_{sc}(w, i, j)$, we have $s \sim_i \overline{s}$ and $\overline{s} \sim_j s'$ where $w = s$, $w' = s'$ and $\overline{s}$ is the added state ($s, \overline{s}, s' \in S$).

Figure 1 illustrates an example of the reduction process from CTLC to CTLK. The reduction of a CTLC formula into a CTLK formula is recursively defined as follows:

- $\mathscr{F}(p) = p$, if p is an atomic proposition.
- $\mathscr{F}(\neg\varphi) = \neg\mathscr{F}(\varphi)$ and $\mathscr{F}(\varphi \vee \psi) = \mathscr{F}(\varphi) \vee \mathscr{F}(\psi)$.
- $\mathscr{F}(\mathtt{EX}\varphi) = \mathtt{EX}\mathscr{F}(\varphi)$ and $\mathscr{F}(\mathtt{E}(\varphi\mathtt{U}\psi)) = \mathtt{E}(\mathscr{F}(\varphi)\mathtt{U}\mathscr{F}(\psi))$.
- $\mathscr{F}(\mathtt{EG}\varphi) = \mathtt{EG}\mathscr{F}(\varphi)$ and $\mathscr{F}(\mathtt{C}(i, j, \varphi)) = \widehat{\mathtt{K}}_i\mathscr{F}(\varphi) \wedge \mathtt{EX}\ \widehat{\mathtt{K}}_j\mathscr{F}(\varphi)$

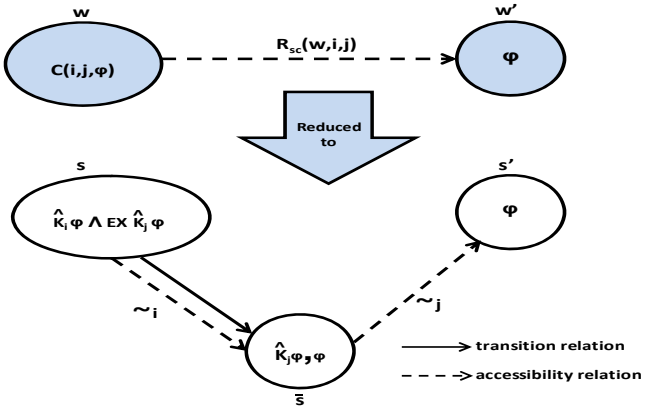

Fig. 1. An example of the reduction process from CTLC to CTLK

Thus, this reduction allows us to model check CTLC formulae by model checking their reductions in CTLK using the MCMAS tool. The most important case in this reduction is the one about commitments (see Fig.1). The following theorem proves the correctness of our reduction from CTLC to CTLK.

Theorem 1 (Correctness). *Let M_{sc} and φ_{sc} be respectively a CTLC model and formula and let $\mathscr{F}(M_{sc})$ and $\mathscr{F}(\varphi_{sc})$ be the corresponding model and formula in CTLK. We have $M_{sc} \models \varphi_{sc}$ iff $\mathscr{F}(M_{sc}) \models \mathscr{F}(\varphi_{sc})$.*

Proof. We prove this theorem by induction on the structure of the formula φ_{sc}:

- If φ_{sc} is a pure CTL formula, the correctness is straightforward from the fact that CTLK is also an extension of CTL.
- If φ_{sc} is not a pure CTL formula, by induction over the structure of φ_{sc}, all the cases are straightforward once the case where $\varphi_{sc} = \mathtt{C}(i, j, \psi)$ is analyzed. In this case we have: $M_{sc}, \langle w \rangle \models \mathtt{C}(i, j, \psi)$ iff for all $w' \in R_{sc}(w, i, j)$ we have $M_{sc}, \langle w' \rangle \models \psi$.
  According to the definition of R_{sc}, we obtain: $M_{sc}, \langle w \rangle \models \mathtt{C}(i, j, \psi)$ iff for all w' such that there exists $\overline{w}$ and $l_i(w) = l_i(\overline{w})$ and $l_j(\overline{w}) = l_j(w')$ we have $M_{sc}, \langle w' \rangle \models \psi$.
  Since $l_j(\overline{s}) = l_j(w')$ and $V(\overline{s}) = V(w')$, we obtain: $\mathscr{F}(M_{sc}), \langle \overline{s} \rangle \models \mathscr{F}(\psi)$ and $\mathscr{F}(M_{sc}), \langle s' \rangle \models \mathscr{F}(\psi)$ and since $s \sim_i \overline{s}$ and $\overline{s} \sim_j s'$, so according to the semantics of $\widehat{\mathtt{K}}_i \mathscr{F}(\psi)$ and $\widehat{\mathtt{K}}_j \mathscr{F}(\psi)$, we get: $\mathscr{F}(M_{sc}), \langle \overline{s} \rangle \models \widehat{\mathtt{K}}_j \mathscr{F}(\psi)$ and $\mathscr{F}(M_{sc}), \langle s \rangle \models \widehat{\mathtt{K}}_i \mathscr{F}(\psi)$.
  So since $(s, \overline{s}) \in T$, we obtain $\mathscr{F}(M_{sc}), \langle s \rangle \models \widehat{\mathtt{K}}_i \mathscr{F}(\psi) \wedge \mathtt{EX}\ \widehat{\mathtt{K}}_j \mathscr{F}(\psi)$. ■

3.3 Reducing CTLC to ARCTL

Lomuscio et al. [16] have proven that the problem of model checking CTLK can be automatically reduced to the problem of model checking ARCTL. ARCTL

is an extension of CTL with action formulae, so it mixes among state formulae and action formulae. However, it restricts path formulae into paths whose actions satisfy a given action formula. Instead of directly reducing CTLC to ARCTL, we simply use the reduction from CTLK to ARCTL since we already reduced CTLC to CTLK. The reduction from CTLC to ARCTL is then obtained by transitivity (see dash arrow in Fig.2).

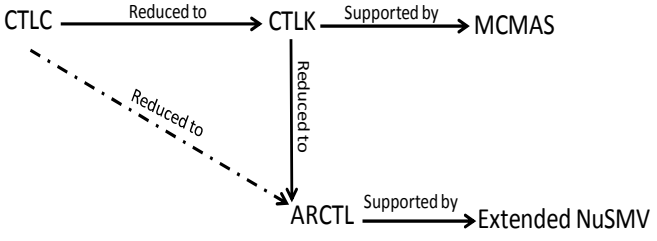

Fig. 2. The reduction processes of CTLC into CTLK and ARCTL

Before we introduce Lomuscio et al.'s reduction technique, we define the syntax of ARCTL using the following BNF grammar [20]:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{E}_\alpha \mathtt{X}\varphi \mid \mathtt{A}_\alpha \mathtt{X}\varphi \mid \mathtt{E}_\alpha(\varphi \mathtt{U} \varphi) \mid \mathtt{A}_\alpha(\varphi \mathtt{U} \varphi)$$
$$\alpha ::= b \mid \neg\alpha \mid \alpha \vee \alpha$$

where φ is state formula, α is action formula, $p \in \Phi_p$ (a set of atomic propositions) and $b \in \Phi_\alpha$ (a set of atomic actions). To define the semantics of ARCTL formulae, the model M is defined as follows: $M = \langle Z, Z_0, A, TR, V_P, V_A \rangle$, where: Z is a set of states; $Z_0 \subseteq Z$ is a set of initial states; A is a set of actions; $TR \subseteq Z \times A \times Z$ is a labeled transition relation; $V_p : Z \rightarrow 2^{\Phi_P}$ is a function that assigns to each state a set of atomic propositions to interpret this state; and $V_A : A \rightarrow 2^{\Phi_\alpha}$ is a function that assigns to each action a set of atomic actions to interpret this action.

The complete semantics of ARCTL is introduced in [20]. The reduction from a CTLK model $M = (S, S_0, T, \{\sim_i\}_{i \in \mathcal{A}}, V)$ to an ARCTL model $M = \langle Z, Z_0, A, TR, V_P, V_A \rangle$ is as follows:

- $Z = S$ and $Z_0 = S_0$.
- reconfiguring the set Φ_α such that $\Phi_\alpha = \{Run, Gt_i, \ldots, Gt_n\}$, where Run is an atomic proposition used to label temporal transitions defined by T and n propositions Gt_i (one for each agent) to label epistemics relations.
- the labeled transition relation TR combines both the temporal transition T and the epistemic relations $\{\sim_i\}_{i \in \mathcal{A}}$ under the following two conditions: for states $s, s' \in S$, (i) $(s, \{Run\}, s') \in TR$ iff $(s, s') \in T$; (ii) $(s, \{Gt_i\}, s') \in TR$ iff $s \sim_i s'$.

The reduction of a CTLK formula into an ARCTL formula is defined as follows [16,20]:

- $\mathscr{F}(p) = p$, if p is an atomic proposition.
- $\mathscr{F}(\neg\varphi) = \neg\mathscr{F}(\varphi)$ and $\mathscr{F}(\varphi \vee \psi) = \mathscr{F}(\varphi) \vee \mathscr{F}(\psi)$.
- $\mathscr{F}(\mathtt{EX}\varphi) = \mathtt{E}_{Run}\mathtt{X}\mathscr{F}(\varphi)$ and $\mathscr{F}(\mathtt{E}(\varphi\mathtt{U}\psi)) = \mathtt{E}_{Run}(\mathscr{F}(\varphi)\mathtt{U}\mathscr{F}(\psi))$.
- $\mathscr{F}(\mathtt{EG}\varphi) = \mathtt{E}_{Run}\mathtt{G}\mathscr{F}(\varphi)$ and $\mathscr{F}(\mathtt{K}_i\varphi) = \mathtt{A}_{Gt_i}\mathtt{X}\mathscr{F}(\varphi)$

Using the reduction from CTLK to ARCTL and our reduction from CTLC to CTLK, we obtain the reduction from CTLC to ARCTL (see Fig.2). However, we can also directly reduce CTLC to ARCTL. The reduction of all CTL formulae is straightforward. The reduction of the commitment formula is as follows: $\mathscr{F}(\mathtt{C}(i,j,\varphi)) = \mathtt{A}_{Gt_i}\mathtt{X}\mathscr{F}(\varphi) \wedge \mathtt{E}_{Run}\mathtt{X}\mathtt{A}_{Gt_j}\mathscr{F}(\varphi)$. The correctness of this reduction follows from Theorem 1 and the correctness of the reduction of CTLK to ARCTL.

4 Commitment Protocols

After reducing CTLC to CTLK and ARCTL, let us apply this reduction to a case study by verifying a commitment protocol. In this section, we define commitment protocols as a set of actions on commitments with respect to the given interpreted system $\mathcal{IS}$. These commitments are defined in our logic CTLC to capture the business interactions among agent roles. In addition to what messages can be exchanged and when, our protocol specifies the meaning of messages in terms of their effects on the commitments. The participating autonomous agents can communicate by exchanging messages in terms of creation and manipulation of commitments such that this exchanging is reliable, meaning that messages do not get lost and the communication channel is order-preserving.

Example 1. We consider the Contract Net Protocol (CNP)[3], as a motivating example to illustrate our representation of commitment protocols. The protocol starts with a manager requesting proposals for a particular task. Each participant either sends a proposal or a reject message. The manager accepts only one proposal among the received proposals and explicitly rejects the rest proposals. The participant with the accepted proposal informs the manager with the proposal result or the failure of the proposal.

Figure 3 depicts our representation of the CNP commitment protocol using commitments and associated actions. It begins with sending a call-for-proposals at state w_0, which means the manager M creates a conditional commitment: $Create(M, P, \mathtt{CC}(M, P, proposal, reply))$ such that if a participant P sends a proposal, the manager will decide and reply with the result of the call-for-proposals (*proposal* and *reply* are *wff* in CTLC). Then, the participant at state w_1 could either accept this call-for-proposal, which means creating a conditional commitment such that if the manager accepts the proposal, the participant will deliver the result of the proposal or reject this call-for-proposal, which means releasing the received commitment and the protocol will achieve the failure state w_3 as a final state. After receiving the participant's proposal, the manager can accept this proposal or reject it.

[3] FIPA Contract Net Interaction Protocol Specification (2002), `http://www.fipa.org/specs/fipa00029/index.html`

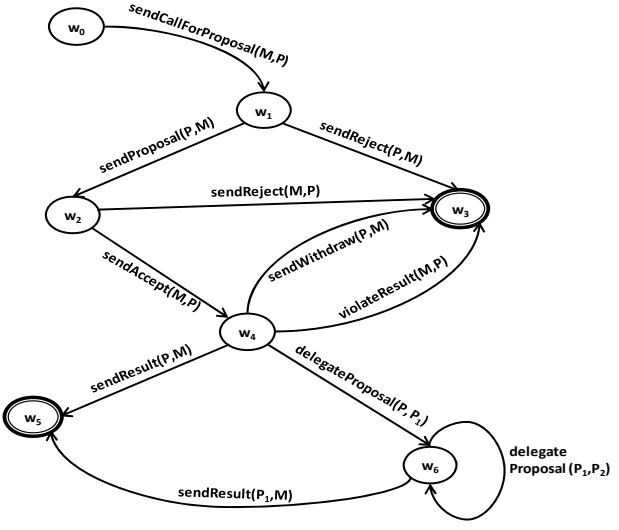

Fig. 3. Contract Net Protocol transitions

By sending the accept message to the participant, the conditional commitment will be transformed to an unconditional commitment at state w_4. At this state, the participant has four possibilities: 1) to withdraw his commitment and then move to the failure state w_3; 2) to delegate it to another participant (say P_1) to deliver the result to the manager on his behalf: $Delegate(P, P_1, \mathtt{C}(P, M, result))$; 3) to violate his commitment and then move to the failure state w_3; or 4) to directly send the result of the proposal to the manager and the protocol will achieve the successful state w_5 as a final state.

As in [28], the participant P_1 can delegate this commitment to another participant (say P_2), which delegates the commitment back to the participant P_1. The participants (P_1 and P_2) delegate the commitment back and forth infinitely often and this is presented by a transition loop at w_6. In a sound protocol, this behavior should be avoided (in Sect.4.2, we will show how to verify this issue). Finally, the participant P_1 can fulfill the delegated commitment by sending the result of the proposal to the manager and then moves to the successful state w_5.

Table 1 depicts the possible actions in the enhanced version of CNP and the corresponding commitment actions.

4.1 Translating Commitment Protocols

The main step in the verification of commitment protocols is translating them into ISPL (the MCMAS's input language) and SMV (the NuSMV's input language). An ISPL program reflects the structure of the interpreted system $\mathcal{IS}$ defined in the following four sections [22]:

Table 1. Actions in the CNP and the corresponding commitment actions

$sendCallForProposal(M, P)$	$Create(M, P, \texttt{CC}(M, P, proposal, reply))$
$sendProposal(P, M)$	$Create(P, M, \texttt{CC}(P, M, accept, result))$
$sendReject(P, M)$	$Release(P, M, \texttt{CC}(M, P, proposal, reply))$
$sendAccept(M, P)$	$Fulfill(M, P, \texttt{C}(M, P, reply))$
$sendWithdraw(P, M)$	$Withdraw(P, M, \texttt{C}(P, M, result))$
$violateResult(P, M)$	$Violate(P, M, \texttt{C}(P, M, result))$
$sendResult(P, M)$	$Fulfill(P, M, \texttt{C}(P, M, result))$
$delegateProposal(P, P_1)$	$Delegate(P, P_1, \texttt{C}(P, M, result))$
$delegateProposal(P_1, P_2)$	$Delegate(P_1, P_2, \texttt{C}(P_1, P, result))$
$sendResult(P_1, M)$	$Fulfill(P_1, M, \texttt{C}(P_1, M, result))$

1. Agents' declarations to define a list of ISPL agents with four sub-sections according to the following syntax: `Agent <agentID> <agent_body> end Agent` where `<agentID>` is an ISPL identifier and `<agent_body>` contains: 1) local states; 2) local actions; 3) local protocol; and 4) evolution function.
2. Evaluation function is defined as follows: `Evaluation <proposition> if <condition_on_states> end Evaluation` where `<proposition>` is an ISPL proposition and `<condition_on_states>` is a truth condition that defines a set of global states for atomic proposition.
3. Initial states to define the set of initial state conditions as follows: `InitStates <condition_on_states> end InitStates`
4. List of formulae needed to be verified is defined using the following syntax: `Formulae <formulae_list> end Formulae`

Our translation process begins by extracting the set of interacting agents: M, P, P_1 and P_2 in our protocol. For each agent, we define the possible commitment states as knowledge states using state variables in the `Vars` sub-section. These variables are of enumeration type, which also include the successful, and failure states. The local actions on commitments are directly defined using the `Actions` sub-section. Using these states and actions, we define the evolution function in the `Evolution` sub-section that captures the transition relations among states. The translation is completed by declaring a set of enabled actions at each state in the `Protocol` sub-section, a set of initial states in the `InitStates` section, and the list of formulae needed to be verified in the `Formulae` section.

As mentioned, we use the extended version of NuSMV introduced in [16], which also uses the extended version of SMV program to verify the translated ARCTL formulae. In the extended version of SMV, the set of interacting agents (M, P, P_1 and P_2 in our protocol) is defined in isolated modules `MODULE Agent<name>`. Figure 4 shows an example of a typical translation of interacting agents in our protocol into extended SMV modules. These modules are instantiated in the main module with the definition of initial conditions using the `TINIT` statement and the keyword `SPEC` to specify the formulae that need to be

```
MODULE main
        VAR M : Manager(args1,args2);
            P : Participant(args1,agrs2);
        TINIT(...);
        SPEC <formulae_list>;

MODULE Manager(args1,agrs2)
        VAR state: {...};
        IVAR action: {...};
        TINIT(...);
        TRANS(next(action)= case ... esac);
        TTRANS(next(state)= case ... esac);
```

Fig. 4. Example of agent translation into extended SMV module

checked. For each agent, we associate the SMV variables `<v1>, ..., <vn>` using the `VAR` statement to define the agents commitment states plus the successful and failure states. The actions of each agent are represented as input variables in `IVAR` statement. The protocol of each agent is defined as a relation among its local state and action variables in the `TRANS` statement. The labeled transitions between commitment states are encoded using the `TTRANS` statement and an initial condition using the `TINIT` statement. Internally, `TTRANS` statements expand to standard `TRANS` statements conditioned on $\{Run\}$ with the `next` and `Case` expressions that represent agent's choices in a sequential manner.

4.2 Protocol Properties

To achieve the flexibility that gives each agent a great freedom and compliance within the same framework, we need to verify the commitment protocols against some properties that capture important requirements in MASs. Specifically, Guerin et al. [14] have proposed three types of verification of multi-agent interaction protocols depending on whether the verification process is done at either design time or run time: 1) verify that an agent will always comply; 2) verify compliance by observation; and 3) verify protocol properties. We adopt the third type of verification for three reasons:

1. The desirable properties play an important role in verifying multi-agent interaction protocols [1,19], which reduces the cost of development process at design time and restricts agents' behaviors by removing bad behaviors without loosing the flexibility.
2. Verifying the compliance of multi-agent interaction protocols with specifications requires adding planner mechanisms equipped with reasoning rules in the code of each agent to reason about its actions to select appropriate ones that satisfy its goals at run time, which can be expensive and may increase the code of the agents [28,24].
3. Protocol properties have a classification in both reactive and distributed systems to guide protocol designers to check protocol specifications.

Some proposals have been put forward to formally express commitment protocol properties [5,10,26]. However, these proposals do not use a specific methodology

to classify protocol properties. Hereafter, we use the classification introduced in [15] to classify temporal properties into: *Safety* and *Liveness*. Manna and Pnueli, in their seminal book [18] have extended the liveness property into: *Guarantee, Obligation, Response, Persistence* and *Reactivity*. In the following, the reachability, deadlock freedom, safety, liveness, and fairness constraint properties are temporal CTLC formulae that we use to check the CNP commitment protocol. Notice that the reachability property do the same function as guarantee property, fairness constraint property captures response and reactivity properties, and obligation property can be defined as a conjunction of safety and reachability properties. Moreover, we omit persistence property as it is mainly related to concurrent behaviors. Consequently, our temporal protocol properties include the properties introduced in [5,10,19] and satisfy the same functionalities of the properties presented in [27].

`Reachability property`. Given a particular state, is there a valid computation sequences to reach that state from an initial state. For example, in all paths in the future (F)[4], there is a possibility for the participant P to deliver the result of the proposal to the manager:

$$\varphi_1 = AFEF\ \mathtt{CC}(M, P, proposal, reply)$$

`Deadlock property`. It is the negation of the reachability property, which is supposed to be false:

$$\varphi_2 = \neg AFEF\ \mathtt{CC}(M, P, proposal, reply)$$

`Fairness constraint property`. The motivation behind this property is to rule out unwanted behaviors of agents and remove any infinite loop in our protocol. For example, if we define the formula:

$$\varphi_3 = AGAF\ \neg\mathtt{C}(P_1, P_2, result)$$

as an unconditional fairness constraint, then a path is fair iff in all paths and in each state of these paths, in all emerging paths P_1 eventually does not delegate commitments. This constraint will enable us to avoid situations such as the participants delegate the commitment back and forth infinitely many times.

`Safety property`. This property means "something bad never happens". For example, in our protocol a bad situation is: the manager sends accept message, but the participant never delivers the result of the proposal:

$$\varphi_4 = AG(\neg\mathtt{C}(M, P, reply)\ \wedge AG\ \neg\mathtt{C}(P, M, result))$$

Liveness: means that "something good will eventually happen". For example, in all paths globally if the manager sends call-for-proposal, then there is a path in the future the participant will send proposal to the manager:

$$\varphi_5 = AG(\mathtt{CC}(M, P, proposal, reply) \rightarrow EF\ \mathtt{CC}(P, M, accept, result))$$

The above formulae are only some examples in our language.

[4] $\mathtt{EF}p$ is the abbreviation of $\mathtt{E}(true\ \mathtt{U}\ p)$.

5 Experimental Results

We implemented the reduction tools on top of the two model checkers (MCMAS and extended NuSMV) and provided a thorough assessment of this reduction on two experiments. In the first experiment, we only consider two party actions on commitments. In the second one, we add more commitments' states by including three party actions on commitments. These experiments were meant to check the effectiveness of our reductions using MCMAS and extended NuSMV in terms of memory in use. They are performed on a laptop with running Windows XP SP2 and equipped with 2.20 GHz AMD Dual Core and 896MB of RAM.

Table 2. Verification results for CNP protocol

	First Experiment		**Second Experiment**	
	Extended NuSMV	MCMAS	Extended NuSMV	MCMAS
Model Size $\|M\|$	$\approx 10^{12}$	$\approx 10^{16}$	$\approx 10^{14}$	$\approx 10^{26}$
Memory in MB	≈ 4.77	≈ 6.37	≈ 4.77	≈ 6.53
# OBDD variables	21	27	23	44
# OBDD nodes	1,241	2,905	1,494	11,885
# agents	2	2	4	4

Table 2 depicts that there is no big difference in the results of extended NuSMV in the two experiments, but by adding three party actions, the number of OBDD variables and nodes in MCMAS are increased. Moreover, the number of OBDD variables and memory size increase by augmenting the number of agents from 2 to 4. The performance of model checker tools also depend on the size of the model M which we define as $|M| = |W| + |R_t|$, where $|W|$ is the number of possible combinations of the states and actions and $|R_t|$ is the temporal relation. In the first experiment, the number of OBDD variables with extended NuSMV (resp. MCMAS) is 21 (resp. 27), then the total state space $|W|$ is $2^{21} \approx 10^6$ (resp. $2^{27} \approx 10^8$). Whereas, in the second experiment, the total state space $|W|$ is $2^{23} \approx 10^7$ in extended NuSMV and $2^{44} \approx 10^{13}$ in MCMAS. We approximate $|R_t|$ as $|W|^2$, hence $|M| = |W| + |R_t| \approx |W|^2$ (see Table 2).

6 Related Work

Several proposals on using existing model checkers (e.g., SPIN and CWB-NC) by translating some agent specification languages (e.g., AgentSpeak(F)) into the languages used by these model checkers [4,5,10,1] have been put forward. In particular, Bordini et al. [4] have introduced the language AgentSpeak(F) and shown how the verification of this language can be translated to the verification

of PROMELA code (the input language of the model checker SPIN). Bentahar et al. [1] have introduced the translation of ACTL* formulae into a variant of alternating tree automata called alternating Büchi tableau automata. Our approach follows the same line of research but it is based on symbolic model checking and not on automata-based model checking like SPIN. Consequently, our approach does not suffer from the state explosion problem, which is a common problem in the automata-based technique. Other researchers have proposed new algorithms for verifying temporal and epistemic properties, see for example [21]. In particular, Lomuscio et al. [17] have proposed MCMAS model checker to verify multi-agent systems based on binary encoding in terms of OBDD representations where properties are specified by means of epistemic modalities such as knowledge modality. This paper shows how high level interactions represented by social commitments can be translated to agents' knowledge without loosing social or public features that characterize commitments.

Recently, Viganò and Colombetti [26] have used symbolic model checking to verify institutions formally modeled with FIEVeL language in terms of the notion of "status function" where properties are specified in an ordered many-sorted first-order temporal logic (OMSFOTL). Their automatic verification process is mainly concerned with satisfying certain properties to guarantee the soundness of institutions without considering any standard temporal properties classification. They regulate interactions between agents in terms of deontic norms (e.g., obligations) that are captured with respect to institution structures. Thus, this model is less flexible than ours as, for example, they do not have possibilities to withdraw or delegate obligations. Gerard and Singh [13] have used CTL and MCMAS to verify protocol refinement that are defined in terms of social commitments without checking the conformance of protocols themselves before the refinement and without considering transition loop within protocol specifications. In terms of commitment protocol properties, Yolum [27] has presented the main generic properties that are required to develop commitment protocols at design time. These properties are categorized into three classes: *effectiveness, consistency* and *robustness*. Our properties meet the same functionalities, for example the reachability and deadlock-freedom can be used to satisfy the same objective of the effectiveness property.

7 Conclusion and Future Work

In this paper, we presented a new language CTLC to represent and reason about social commitments. We used this language to specify commitment protocols and their temporal properties in electronic business domains. We showed how to reduce the problem of model checking CTLC to the problem of model checking either CTLK or ARCTL. Thus, it is the first step towards achieving the following features within the same framework that formalizes commitment protocols: 1) formal (based on our logic); 2) meaningful (in terms of social commitments); 3) declarative (which focuses on what the message means not how the message is exchanged); 4) verifiable (using efficient and available symbolic model

checking); and 5) property-based (in terms of formally defined properties). To clarify our approach, we have modeled the Contract Net Protocol (CNP) using commitments and associated actions. In our implementation, we conducted two experiments, which revealed promising results for multi-agent systems where interaction protocols are involved. There are many directions for future work. We plan to expand the formalization of commitment protocols with metacommitments. We also plan to investigate other reductions, particularly from CTL^{*c} (an extension of CTL* with commitment modality) to GCTL* (generalized CTL*) [3], so that we can use the CWB-NC model checker.

Acknowledgements

We would like to thank the reviewers for their valuable comments and suggestions. Jamal Bentahar and Rachida Dssouli would like to thank Natural Sciences and Engineering Research Council of Canada (NSERC) and Fond Québecois de la recherche sur la société et la culture (FQRSC) for their financial support.

References

1. Bentahar, J., Meyer, J.J.C., Wan, W.: Model Checking Agent Communication. In: Dastani, M., Hindriks, K.V., Meyer, J.J.C. (eds.) Specification and Verification of Multi-Agent Systems, 1st edn., pp. 67–102. Springer, Heidelberg (2010)
2. Bentahar, J., Moulin, B., Chaib-draa, B.: Specifying and Implementing a Persuasion Dialogue Game using Commitment and Argument Network. In: Rahwan, I., Moraïtis, P., Reed, C. (eds.) ArgMAS 2004. LNCS (LNAI), vol. 3366, pp. 130–148. Springer, Heidelberg (2005)
3. Bhat, G., Cleaveland, R., Groce, A.: Efficient Model Checking via Büchi Tableau Automata. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 38–52. Springer, Heidelberg (2001)
4. Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model Checking Agentspeak. In: Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 409–416. ACM, Melbourne (2003)
5. Cheng, Z.: Verifying Commitment based Business Protocols and their Compositions: Model Checking using Promela and Spin. Ph.D. thesis, North Carolina State University (2006)
6. Chopra, A.K., Singh, M.P.: Nonmonotonic Commitment Machines. In: Dignum, F. (ed.) ACL 2003. LNCS (LNAI), vol. 2922, pp. 183–200. Springer, Heidelberg (2004)
7. Chopra, A.K., Singh, M.P.: Producing Compliant Interactions: Conformance, Coverage and Interoperability. In: Baldoni, M., Endriss, U. (eds.) DALT IV 2006. LNCS (LNAI), vol. 4327, pp. 1–15. Springer, Heidelberg (2006)
8. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An Open Source Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)

9. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
10. Desai, N., Cheng, Z., Chopra, A.K., Singh, M.P.: Toward Verification of Commitment Protocols and their Compositions. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 144–146. ACM, Honolulu (2007)
11. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. The MIT Press, Cambridge (1995)
12. Fornara, N., Colombetti, M.: Operational Specification of a Commitment-based Agent Communication Language. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 535–542. ACM, Bologna (2002)
13. Gerard, S.N., Singh, M.P.: Protocol Refinement: Formalization and Verification. In: Artikis, A., Bentahar, J., Chopra, A.K., Dignum, F. (eds.) AAMAS Workshop on Agent Communication (AC), Toronto, Canada, pp. 19–36 (2010)
14. Guerin, F., Pitt, J.: Guaranteeing Properties for E-Commerce Systems. In: Padget, J.A., Shehory, O., Parkes, D.C., Sadeh, N.M., Walsh, W.E. (eds.) AMEC IV 2002. LNCS (LNAI), vol. 2531, pp. 253–272. Springer, Heidelberg (2002)
15. Lamport, L.: Proving the Correctness of Multiprocess Programs. IEEE Transactions on Software Engineering 3(2), 125–143 (1977)
16. Lomuscio, A., Pecheur, C., Raimondi, F.: Automatic Verification of Knowledge and Time with Nusmv. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 1384–1389. Morgan Kaufmann Publishers Inc., Hyderabad (2007)
17. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 682–688. Springer, Heidelberg (2009)
18. Manna, Z., Pnueli, A.: The Temporal Logic of Rreactive and Concurrent Systems: Specification, 1st edn. Springer, Inc., New York (1991)
19. Medellin, R., Atkinson, K., McBurney, P.: Model Checking Command Dialogues. In: Proceedings of 2009 AAAI Fall Symposium on The Uses of Computational Argumentation, pp. 58–63. AAAI Press, Arlington (2009)
20. Pecheur, C., Raimondi, F.: Symbolic model checking of logics with actions. In: Edelkamp, S., Lomuscio, A. (eds.) MoChArt IV. LNCS (LNAI), vol. 4428, pp. 113–128. Springer, Heidelberg (2007)
21. Penczek, W., Lomuscio, A.: Verifying Epistemic Properties of Multi-Agent Systems via Bounded Model Checking. Fundamenta Informaticae 55(2), 167–185 (2003)
22. Raimondi, F.: Model Checking Multi-Agent Systems. Ph.D. thesis, University College London (2006)
23. Singh, M.P.: An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. Artificial Intelligent and Law 7(1), 97–113 (1999)
24. Singh, M.P.: Formalizing Communication Protocols for Multiagent Systems. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 1519–1524. Morgan Kaufmann Publishers, Inc., Hyderabad (2007)
25. Venkatraman, M., Singh, M.P.: Verifying Compliance with Commitment Protocols: Enabling Open Web-based Multiagent Systems. Autonomous Agents and Multi-Agent Systems 2(3), 217–236 (1999)

26. Viganò, F., Colombetti, M.: Symbolic Model Checking of Institutions. In: Proceedings of the 9th International Conference on Electronic Commerce, pp. 35–44. ACM Press, Minneapolis (2007)
27. Yolum, P.: Design Time Analysis of Multi-Agent Protocols. Data and Knowladge Engineering 63(1), 137–1154 (2007)
28. Yolum, P., Singh, M.P.: Commitment machines. In: Meyer, J.J.C., Tambe, M. (eds.) ATAL 2001. LNCS (LNAI), vol. 2333, pp. 235–247. Springer, Heidelberg (2002)
29. Yolum, P., Singh, M.P.: Reasoning about Commitments in the Event Calculus: An Approach for Sepcifying and Executing Protocols. Annals of Mathematics and Artificial Intelligence 42(1-3), 227–253 (2004)