Model Checking Commitment Protocols

Mohamed El-Menshawy, Jamal Bentahar, and Rachida Dssouli

Concordia University, Faculty of Engineering and Computer Science, Canada
m_elme@encs.concordia.ca, bentahar@ciise.concordia.ca,
dssouli@ece.concordia.ca

Abstract. We investigate the problem of verifying commitment protocols that are widely used to regulate interactions among cognitive agents by means of model checking. We present a new logic-based language to specify commitment protocols, which is derived from extending CTL* with modalities for social commitments and associated actions. We report on the implementation of the NetBill protocol—a motivated and specified example in the proposed language—using three model checkers (MCMAS, NuSMV, and CWB-NC) and compare the experimental results obtained.

1 Introduction

A society mainly evolves through communication among cognitive participating entities. In such a society, people interact and exchange information with each other to satisfy their goals. Such communication requires languages and mechanisms to structure interactions among participants within dialogues.

Correspondingly, a cognitive agent-based model for an artificial society should provide adequate support for communication languages by providing means to specify interaction protocols that regulate intelligent interactions among agents. It is also beneficial to equip this model with a formal verification method to eliminate errors in the sense that protocols comply with the given specifications and to increase confidence on the model in terms of safety and efficiency.

Motivation. This paper introduces a new logic-based language to specify a special kind of interaction protocols, called commitment protocols [5,6,13,12]. This logic extends CTL* [4] with modalities for social commitments and associated actions. Social commitments usually represent the contractual obligations that associate the cognitive agents with one another in order to capture multi-agent intelligent interactions in this artificial society. In this context, social commitments model cognitive aspects of agent interactions by enforcing them to keep the fulfillment of their contracts and reducing cognitive dissonance in agent behaviors. Moreover, we adopt these commitment protocols as they are increasingly used in different applications such as modeling business processes [5,6,13] and developing web-based applications [11,2,7]. Conventionally, commitment protocols are defined in terms of creation and manipulation of social commitments that capture a rich variety of interactions among agents. Several approaches have

K.G. Mehrotra et al. (Eds.): IEA/AIE 2011, Part II, LNAI 6704, pp. 37–47, 2011.

been proposed to specify commitment protocols [6,13,12,11]. However, the languages used to specify those protocols are either not suitable for model checking (a formal verification method) or defined in computational logics having weak expressive power.

Approach. Our approach begins with developing a new logic-based language by extending CTL* with modalities for social commitments and associated actions. The resulting logic, ACTL^{*sc}, is used to: 1) express well-formed formulae of commitments and their contents; 2) specify an abstract semantics of commitment actions; and 3) express desirable properties to be checked such as safety.

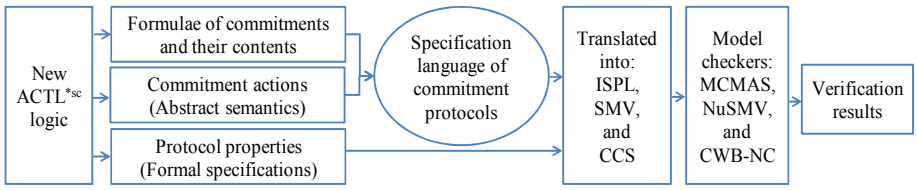

Fig. 1. A schematic view of our approach

By abstract semantics, we mean a semantics that captures the performance of the commitment actions without defining the meaning of all concrete action instances. Using an abstract action makes our approach more flexible as it allows for capturing not only standard commitment actions like those introduced in [10] such as Withdraw, but also other possible actions such as Inform. However, the concrete semantics of the commitment actions considered in this paper is completely given in our previous work [8]. Using social commitments and their actions, we define a new specification language of commitment protocols. We present experimental results of verifying the NetBill protocol taken from e-business domain and specified in our language against some properties. Finally, the implementation of this protocol is done using the MCMAS model checker [9] along with NuSMV [3] and CWB-NC [14] as benchmarks after translating the protocol into these model checkers input languages ISPL, SMV and CCS respectively (see Figure 1).

Organization. Section 2 presents the syntax and semantics of ACTL^{*sc} logic. In Section 3, we use commitments and their actions to define a new specification language of commitment protocols. The translation and implementation of the NetBill protocol and its verification using the MCMAS, NuSMV and CWB-NC model checkers with experimental results are discussed in Section 4. In Section 5, we discuss relevant literature and future work.

2 Commitments and ACTL^{*sc} Logic

The introduction of a new branching time ACTL^{*sc} logic is motivated by the fact that the needed modal connectives for social commitments and associated

actions cannot be only expressed using CTL* [4]. Before introducing the syntax and semantics of our logic, we present the notion of social commitments.

Notation 1. *Social commitments are denoted by $SC(i, j, \varphi)$, where i is the debtor, j the creditor and φ a well-formed formula (wff) in the proposed logic representing the commitment content. $SC(i, j, \varphi)$ means i socially (i.e., publicly) commits to j that φ holds [10,8].*

Notation 2. *Conditional commitments are denoted by $\tau \rightarrow SC(i, j, \varphi)$, where "$\rightarrow$" is the logical implication, i, j and φ have the above meanings and τ is a wff in the proposed logic representing the commitment condition [10,8].*

We use $SC^c(i, j, \tau, \varphi)$ as an abbreviation of $\tau \rightarrow SC(i, j, \varphi)$. In this case, we have $SC(i, j, \varphi) \triangleq SC^c(i, j, \top, \varphi)$, where $\top$ is the constant for truth. Commitments can be manipulated using a set actions defined in [10]. We classify these actions into two and three party actions. The former ones need only two agents to be performed such as: $\mathsf{Create}(i, j, SC(i, j, \varphi))$ to establish a new commitment, $\mathsf{Withdraw}(i, j, SC(i, j, \varphi))$ to cancel an existing commitment, $\mathsf{Fulfill}(i, j, SC(i, j, \varphi))$ to satisfy the commitment content, $\mathsf{Violate}(i, j, SC(i, j, \varphi))$ to reflect there is no way to satisfy the commitment content, and $\mathsf{Release}(j, i, SC(i, j, \varphi))$ to free a debtor from carrying out his commitment. The latter ones need an intermediate agent to be completed such as: $\mathsf{Delegate}(i, k, SC(i, j, \varphi))$ to delegate an existing commitment to another debtor k to satisfy it on his behalf and $\mathsf{Assign}(j, k, SC(i, j, \varphi))$ to assign an existing commitment to another creditor k.

2.1 Syntax of ACTL^{*sc}

We use $\Phi_p = \{p, p_1, p_2, \ldots\}$ for a set of atomic propositions, $\mathtt{AGT} = \{i, j, k, \ldots\}$ for a set of agent names and $\mathtt{ACT} = \{\theta, \theta_1, \theta_2, \ldots\}$ for a set of actions.

Definition 1. *The syntax of ACTL^{*sc} is given by the following BNF grammar:*

$$\mathcal{S} ::= p \mid \neg\mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid E\ \mathcal{P} \mid \mathcal{C}$$
$$\mathcal{C} ::= SC(Agt, Agt, \mathcal{P})$$
$$\mathcal{P} ::= \mathcal{S} \mid \neg\mathcal{P} \mid \mathcal{P} \vee \mathcal{P} \mid \mathsf{X}\mathcal{P} \mid \mathcal{P}\,\mathsf{U}\,\mathcal{P} \mid Create(Agt, Agt, \mathcal{C}) \mid \Theta(Agt, Agt, \mathcal{C})$$

where Agt and Θ are non-terminals corresponding to `AGT` and `ACT` respectively. Formulae in ACTL^{*sc} are classified into state formulae $\mathcal{S}$ and path formulae $\mathcal{P}$. The intuitive meanings of the most constructs of ACTL^{*sc} are straightforward from CTL* operators. The formula $A\varphi$ (resp. $E\varphi$) means that φ holds along all (some) paths starting at the current state. The formula $SC(i, j, \varphi)$ means agent i commits towards agent j to bring about a formula φ. The action formula $\theta(i, j, \mathcal{C})$ means an action θ is performed on the commitment $\mathcal{C}$ by i towards j. Furthermore, there are some useful abbreviations based on temporal operators: $F\varphi \triangleq true\ \mathsf{U}\varphi$ (sometimes in the future) and $G\varphi \triangleq \neg F\neg\varphi$ (globally).

2.2 Semantics of ACTL^{*sc}

The semantics of ACTL^{*sc} logic is interpreted with respect to the formal model M associated to the commitment protocol using a Kripke structure as follows: $M = \langle \mathbb{S}, \mathtt{ACT}, \mathtt{AGT}, R, \mathbb{V}, \mathbb{R}_{sc}, \mathbb{L}, \mathtt{AS}, I \rangle$, where: $\mathbb{S}$ is a finite set of states; $\mathtt{AGT}$ is a set of agents; $\mathtt{ACT}$ is a set of actions; $R \subseteq \mathbb{S} \times \mathtt{AGT} \times \mathtt{ACT} \times \mathbb{S}$ is a total labeled transition relation; $\mathbb{V} : \Phi_p \rightarrow 2^{\mathbb{S}}$ is a valuation function; $\mathbb{R}_{sc} : \mathbb{S} \times \mathtt{AGT} \times \mathtt{AGT} \rightarrow 2^{\Pi}$, where Π is the set of all paths, is the accessibility relation for social commitments; $\mathbb{L} : \mathbb{S} \rightarrow 2^{\mathtt{AGT} \times \mathtt{AGT}}$ is an agency function that associates to each state a set of two interacting agents in this state; $\mathtt{AS} \subseteq \mathbb{S}$ is a set of acceptance states; and $I \subseteq \mathbb{S}$ is a set of initial states.

Instead of $(s_k, Ag_l, \theta_k, s_{k+1})$, the labeled transitions will be written as $s_k \xrightarrow{Ag_l:\theta_k} s_{k+1}$. The paths that path formulae are interpreted over have the form $\pi = \langle s_k \xrightarrow{Ag_l:\theta_k} s_{k+1} \xrightarrow{Ag_{l+1}:\theta_{k+1}} s_{k+2} \ldots \rangle$ such that $\forall k, l \geq 0, (s_k \xrightarrow{Ag_l:\theta_k} s_{k+1}) \in R$. A path in M is then an infinite sequence of states and labeled transitions with agents and their actions. $\pi(m)$ refers to the m^{th} state in this sequence. When a state s_m is a part of a path π, we write $s_m \in \pi$. The set of all paths starting at s_k is denoted by Π^{s_k} and $\langle s_k, \pi \rangle$ refers to the path π starting at s_k. A path $\pi \in \mathbb{R}_{sc}(s_k, i, j)$ is an accessible path for the interacting agents i and j iff all states along this path are possible states for the two interacting agents. Formally: $\pi \in \mathbb{R}_{sc}(s_k, i, j)$ iff $s_k = \pi(0)$ and $\forall s_m \in \pi$ we have $(i, j) \in \mathbb{L}(s_m)$.

Excluding social commitments and action formulae, the semantics of ACTL^{*sc} state formulae is as usual (semantics of CTL*). The notation $M, \langle s_k \rangle \models \varphi$ means that the model M satisfies φ at state s_k and $M, \langle s_k, \pi \rangle \models \varphi$ means that the model M satisfies φ along the path π starting at state s_k ($s_k = \pi(0)$). The state formula $SC(i, j, \varphi)$ is satisfied in the model M at s_k iff the content φ is true in every accessible path from this state using $\mathbb{R}_{sc}(s_k, i, j)$.

$M, \langle s_k \rangle \models SC(i, j, \varphi)$ iff $\forall \pi \in \Pi^{s_k}$ s.t. $\pi \in \mathbb{R}_{sc}(s_k, i, j)$ we have $M, \langle s_k, \pi \rangle \models \varphi$

A path π emanating from s_k satisfies $\mathsf{Create}(i, j, SC(i, j, \varphi))$ in the model M iff i: Create is the label of the first transition on this path and $SC(i, j, \varphi)$ holds in the next state s_{k+1} on this path.

$M, \langle s_k, \pi \rangle \models \mathsf{Create}(i, j, SC(i, j, \varphi))$ iff $s_k \xrightarrow{i:\mathsf{Create}} s_{k+1} \in R$ and
$M, \langle s_{k+1} \rangle \models SC(i, j, \varphi)$

A path π emanating from s_k satisfies $\theta(Ag_x, Ag_y, SC(i, j, \varphi))$ in the model M where $Ag_x \neq Ag_y$, $x \in \{1, 2\}$ and $y \in \{1, 2, 3\}$ iff $Ag_x : \theta$ is the label of the first transition on this path and the commitment has been created in the past.

$M, \langle s_k, \pi \rangle \models \theta(Ag_x, Ag_y, SC(i, j, \varphi))$ iff $s_k \xrightarrow{Ag_x:\theta} s_{k+1} \in R$ and $\exists l \leq k$ s.t. $M, \langle s_l, \pi \downarrow s_l \rangle \models \mathsf{Create}(i, j, SC(i, j, \varphi))$

where $\pi \downarrow s_l$ is the prefix of π starting at s_l.

3 Specification of Commitment Protocols

Our specification language of commitment protocols is defined as a set of commitments capturing the business interactions among agents (or roles) at design

time. In addition to which messages can be exchanged and when, a protocol also specifies the meanings of the messages in terms of their effects on the commitments and each message can be mapped to an action on a commitment.

The proposed specification language begins with the commitment COM, which is followed by a message MSG. This message is directly mapped into either commitment actions (captured by θ) or inform action. Specifically, MSG could either be Withdraw, Fulfill, Violate, Release, Assign, Delegate or Inform. The delegated (resp. the assigned) message is followed by create message that enables the delegatee (resp. the assignee) to create a new commitment. The inform message is an action performed by the debtor i to inform the creditor j that a domain proposition holds. It is not a commitment action, but indirectly affects commitments by causing transformation from conditional to unconditional commitments. The domain proposition Dom-Pro identifies the set of propositions related to the application domain of the protocol. The protocol terminates when the interacting agents do not have commitments to each other. The formal specification of commitment protocols is defined using BNF grammar with meta-symbols "::=" and "|" for the choice and ";" for action sequence as follows:

Table 1. The formal specification of commitment protocols

Protocol	::= COM ; MSG
MSG	::= Withdraw(i, j, COM) | Fulfill(i, j, COM) | Violate(i, j, COM) | Release(j, i, COM) | [(Assign(j, k, COM) | Delegate(i, k, COM)) ; Create(i, j, COM)] | Inform(i, j, Dom-Pro) ; MSG
COM	::= SC^c(i, j, Prop, Prop) | SC(i, j, Prop)
Prop	::= A well−formed formula in ACTL^{*sc}
Dom-Pro	::= Identify domain propositions

The above specification can either be used to reason about the actions at run time or compiled into finite state machines (FSMs) at design time. At run time, the agents can logically compute their execution paths that respect the given protocol specifications using some reasoning rules as axioms [13]. However, these rules are not enough to verify protocols against some properties when the system is complex. Moreover, the flexibility resulting from these reasoning rules can be expensive and may increase the code of the agents [13]. For these reasons, we adopt the second approach, which consists in compiling the commitment protocol into enhanced FSM wherein commitments hold on states and actions label transitions, and thereby the legal computations can be inferred. This enables the protocols to be enhanced with additional transitions to handle exceptions.

3.1 A Motivating Example

Let us consider the NetBill protocol used in [13,8,1] and taken from e-business domain to clarify our specification of commitment protocols. Figure 2 depicts

our modeling of the NetBill protocol using enhanced FSM where $\mathbb{S}$ is the set of vertices, `AGT` and `ACT` the set of edge labels, $I = s_0$, `AS` $= \{s_6\}$, and R the set of edges. Our modeling begins by an inform message on a path starting at s_0. In this message, the customer (Cus) requests a quote for some goods. This request is followed by the merchant (Mer) reply that consists in sending the quote as an offer, which means creating a commitment. The Cus agent could either reject this offer, which means releasing the offer and the protocol ends at the failure state s_7 (as it is not in the set AS), or accept the offer, which means creating a commitment at s_3. When the Cus agent accepts the received offer, then he has three choices: 1) to withdraw his commitment; 2) to delegate it to a financial company (say $Bank_1$); or 3) to directly pay the Mer agent. As in [12], the $Bank_1$ agent can delegate this commitment to another bank (say $Bank_2$), which delegates the commitment back to the $Bank_1$ agent. The bank agents delegate the commitment back and forth infinitely often and this is presented by a transition loop at s_{11}. In a sound protocol, this behavior should be avoided (see Section 4.2). The Mer agent, before delivering the goods to the Cus agent, can withdraw his offer and then move to the recovery state s_9 after refunding the payment to the Cus agent. When the Cus agent pays for the requested goods and the Mer agent delivers them, the Mer agent fulfills his commitment at s_5 and then moves to the acceptance state s_6 after sending the receipt to the Cus agent. Conversely, the Cus agent can pay for the requested goods without being delivered by the Mer agent. In this case, the merchant agent violates his commitment at s_8 and then moves to the recovery state s_9 after refunding the payment to the customer. Finally, the Mer agent, for some reasons, can assign his commitment to another merchant (say Mer_1) at s_{12} in which a new commitment between Cus and Mer_1 is created as a new offer. As for the delegate scenario, the assign action can be repeated infinitely often among agents and this scenario, presented by a transition loop at s_{12}, is unwanted behavior in our protocol.

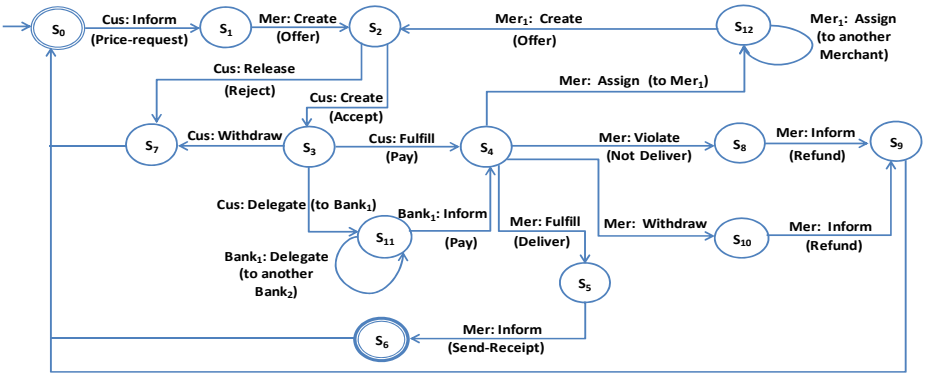

Fig. 2. Enhanced FSM of NetBill protocol with commitments and their actions

4 Implementation

In a nutshell, given the model M representing the NetBill protocol and a formula φ, the problem of model checking is establishing whether or not the model M satisfies φ. Clarke et al. shown that the problem of CTL* model checking can be reduced to the problem of CTL and LTL model checking [4]. This paper follows similar approach by effectively reducing the problem of ACTL^{*sc} model checking to the problem of model checking ALTL^{sc} and ACTL^{sc}. ALTL^{sc} and ACTL^{sc} are LTL and CTL augmented with commitments and action formulae.

4.1 Translating Commitment Protocols

The first step in our implementation is to translate the NetBill protocol into: 1) ISPL (the input language of MCMAS); 2) SMV (the input language of NuSMV); and 3) CCS (the input language of CWB-NC). In ISPL, this process begins by translating a set of interacting agents (Mer, Cus, Mer_1 and $Bank_1$) into standard agents in the `Agent` section and commitments into local variables in the `Vars` section. Such variables are of enumeration type including all possible commitment states plus the acceptance, failure and recovery states. The actions on commitments are directly defined in the `Actions` section. The translation is completed by declaring a set I of initial states in the `InitStates` section and the formulae that need to be checked in the `Formulae` section.

In SMV, the set of interacting agents are translated into isolated modules, which are instantiated in the main module. The latter module includes the definition of initial conditions using the `INIT` statement and the keyword `SPEC` to specify the formulae that need to be checked. The commitment states plus the successful, failure and recovery states are defined as SMV variables in the `VAR` statement. Such states with actions are used as reasoning rules to automatically evolve the states change. The labeled transition relation between commitment states is described using the `TRANS` statement. This statement uses the `next` and `Case` expressions to represent agent's choices in a sequential manner.

In CCS, each agent in our protocol is represented by a set of processes and each process is specified using the `proc` statement. The states of commitments are captured by the set of actions performed on these commitments. Each transition is represented by the action labeling this transition followed by another process.

4.2 Expressing Protocol Properties

To automatically verify commitment protocols, we need to express some desirable properties. Specifically, we formally express a rich class of protocol properties: *fairness*, *safety*, *liveness*, and *reachability* using the proposed logic. These properties include the properties defined in [2,5] and satisfy the same functionalities of the properties defined in [12,1] (see our discussion in Section 5).

`Fairness constraint property.` It is needed to rule out unwanted behaviors of agents in our protocol. An example of fairness constraint is given by φ_1, which

states that along all paths it is globally the case that on all emerging paths the $Bank_1$ agent eventually does not delegate commitment.

$$\varphi_1 = AGAF\neg\mathsf{Delegate}(Bank_1)$$

`Safety property.` This property means "something bad never happens". For example, in our protocol a bad situation is: the Cus agent sends the payment, but the Mer agent never delivers the requested goods:

$$\varphi_2 = AG(\neg(Pay(Cus) \ \wedge AG\neg Deliver(Mer))$$

`Liveness property.` This property means "something good will eventually happen". For example, in our protocol a good situation is: along all paths it is globally the case that if the Cus agent sends the payment, then in all computations in the future the Mer agent will either 1) deliver the requested goods; 2) withdraw his commitment; or 3) violate his commitment:

$$\varphi_3 = AG(Pay(Cus) \rightarrow AF(Deliver(Mer) \vee \mathsf{Withdraw}(Mer) \vee (NotDeliver(Mer)))$$

`Reachability property.` A particular situation can be reached from the initial state via some computation sequences. For example, along all paths it is eventually the case that there is a possibility in the future for the Mer agent to deliver the requested goods to the Cus agent:

$$\varphi_4 = AFEFDeliver(Mer)$$

4.3 Experimental Results

We implemented the translation tools on top of the three model checkers (MCMAS, NuSMV, and CWB-NC) and provided a thorough assessment of this translation on three experiments. These experiments were meant to compare the verification results of checking the NetBill protocol against some properties specified in ACTL^{sc} (fragment of our logic) using MCMAS and NuSMV (symbolic-based techniques) and CWB-NC (an automata-based technique). We used two criteria (the model size and execution time) to perform these comparisons. The model size is defined as $|M| = |\mathbb{S}| + |R|$, where $|\mathbb{S}|$ is the state space and $|R|$ is the relation space. The state space is computed by multiplying the state space of each agent and the state space of the protocol. We can approximate $|R|$ by $10\text{x}|\mathbb{S}|^2$, where 10 is the number of ACTL^{*sc} operators. So we have $|M| \approx 10\text{x}|\mathbb{S}|^2$. The model size of our experiments is shown in Table 2.

Table 3 displays the full results of these three experiments and the execution times (in seconds) on a laptop running 32-bit Windows Vista with 2 GiB of memory and Intel Core 2 Duo clocked at 1.66 GHz. For instance, the execution time for MCMAS increases when augmenting the number of agents from 2 to 4 as the number of OBDD variables needed to encode agents increases with

Table 2. The model size in the three experiments

	Exp.1	**Exp.2**	**Exp.3**
Model size	3.5E+7	9.73E+10	6.23E+11

the number of agents. This proves the fact that the time complexity of model checking is a function of the size of the input problem. The results also reveal that the number of OBDD variables (which reflect the model size) is greater in NuSMV than in MCMAS, but the execution time in NuSMV is better than in MCMAS. In CWB-NC, when the size of the model is small, the execution time is better than in MCMAS and NuSMV (see Table 3). However, when the state space increases (as in Exp.3), the execution time in MCMAS and NuSMV is going to be better than in CWB-NC. Notice that we put "–" in Table 3 because CWB-NC does not use OBDD variables. We can conclude that MCMAS and NuSMV are better than CWB-NC in terms of the execution time.

Table 3. Verification results of the MCMAS, NuSMV and CWB-NC

	MCMAS			**NuSMV**			**CWB-NC**		
	Exp.1	**Exp.2**	**Exp.3**	**Exp.1**	**Exp.2**	**Exp.3**	**Exp.1**	**Exp.2**	**Exp.3**
# OBDDs	24	39	49	33	53	67	–	–	–
# Agents	2	3	4	2	3	4	2	3	4
Exec. Time (sec)	0.52	2	6	0.23	1.11	1.98	0.094	0.564	8.814

5 Related Literature and Conclusion

Desai et al. [6] used a modular action description language to specify commitment protocols. Their approach composes multiple protocols to simplify the development of business processes. Yolum and Singh [13] used commitment actions to show how to build and reason about commitment protocols using event calculus. Their approach only indirectly models the fulfillment of a commitment. Our approach belongs to the same line of research, but it complements the above frameworks by defining a more expressive logic-based language to specify commitment protocols, which is suitable for model checking.

Venkatraman and Singh [11] presented an approach for testing whether the behavior of an agent complies with a commitment protocol specified in CTL. The proposed approach uses model checking to automatically verify the conformance of commitment protocols with specifications (properties). Cheng [2] and Desai et al. [5] used the automata-based model checker SPIN to verify commitment business protocols and their compositions expressed in LTL. Our language uses not only ALTL^{sc} specification, but also ACTL^{sc} specification. We also used the MCMAS and NuSMV tools, which are more efficient than SPIN (see Table 3). Bentahar et al. [1] proposed a framework where the interacting agent-based systems communicate by combing and reasoning about dialogue games. Our verification technique is based on encoding the protocol and formulae using symbolic model checkers instead of translating them into alternating Büchi tableau automata. Yolum [12] presented the main generic properties needed to design commitment protocols. Our properties meet these generic properties in the sense that the reachability can be used to satisfy the same objective of the effectiveness property. The consistency property is achieved in our protocol by

satisfying the safety property. Moreover, the robustness property is satisfied by considering liveness property and fairness paths.

Conclusion. In this paper, we presented a new logic to define a specification language of commitment protocols, which excludes spurious aspects that plague existing approaches. We described a verification technique to automatically—instead of semi-automatically [12]—verify the compliance of these protocols with the given properties. In our implementation, we conducted 3 experiments, which demonstrate the following fact: three model checkers were able to verify a variety of complex formulae correctly and efficiently but the symbolic technique is better than the automata one. As future work, we plan to develop a new model checking algorithm for our logic to directly verify the proposed specification language.

Acknowledgements. The authors 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. Cheng, Z.: Verifying Commitment based Business Protocols and their Compositions: Model Checking using Promela and Spin. Ph.D. thesis, North Carolina State University (2006)
3. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV: 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)
4. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
5. Desai, N., Cheng, Z., Chopra, A.K., Singh, M.P.: Toward Verification of Commitment Protocols and their Compositions. In: Durfee, E.H., Yokoo, M., Huhns, M.N., Shehory, O. (eds.) AAMAS, pp. 144–146. IFAAMAS (2007)
6. Desai, N., Singh, M.P.: A Modular Action Description Language for Protocol Composition. In: Proc. of the 22th AAAI Conference on AI, pp. 962–967 (2007)
7. El-Menshawy, M., Bentahar, J., Dssouli, R.: Modeling and verifying business interactions via commitments and dialogue actions. In: Jędrzejowicz, P., Nguyen, N.T., Howlet, R.J., Jain, L.C. (eds.) KES-AMSTA 2010. LNCS, vol. 6071, pp. 11–21. Springer, Heidelberg (2010)
8. El-Menshawy, M., Bentahar, J., Dssouli, R.: Verifiable semantic model for agent interactions using social commitments. In: Dastani, M., El Fallah Segrouchni, A., Leite, J., Torroni, P. (eds.) LADS 2009. LNCS, vol. 6039, pp. 128–152. Springer, Heidelberg (2010)
9. 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)

10. Singh, M.P.: An Ontology for Commitments in Multi-Agent Systems: Toward a Unification of Normative Concepts. AI and Law 7(1), 97–113 (1999)
11. Venkatraman, M., Singh, M.P.: Verifying Compliance with Commitment Protocols: Enabling Open Web-based Multi-Agent Systems. Autonomous Agents and Multi-Agent Systems 2(3), 217–236 (1999)
12. Yolum, P.: Design Time Analysis of Multi-Agent Protocols. Data and Knowladge Engineering 63(1), 137–154 (2007)
13. Yolum, P., Singh, M.P.: Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitment. In: Proc. of the 1st International Joint Conference on AAMAS, pp. 527–534. ACM, New York (2002)
14. Zhang, D., Cleaveland, W.R., Stark, E.W.: The integrated CWB-nC/PIOATool for functional verification and performance analysis of concurrent systems. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 431–436. Springer, Heidelberg (2003)