Verifying Conformance of Commitment Protocols via Symbolic Model Checking

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

[1] Concordia University, Department of Electrical and Computer Engineering Canada
{m_elme,w_wan}@encs.concordia.ca
[2] Concordia University, Concordia Institute for Information Systems Eng., Canada
{bentahar,dssouli}@ciise.concordia.ca

Abstract. Commitment protocols have been widely used to capture flexible and rich interactions among agents in multi-agent systems. Although there are several approaches specifying commitment protocols, none of them synthesize formal specification and automatic verification of these protocols within the same framework. This paper presents a new approach to automatically verify the conformance of commitment protocols having a social semantics with specifications at design time. The contributions of this paper are twofold: first, we present a new language to formally specify the commitment protocols, which is derived from a new logic extending CTL^* with modality of social commitments and actions on these commitments; and second, we develop a symbolic model checking algorithm for the proposed logic, which is used to express the protocol properties we aim to check such as safety and liveness. We also present experimental results of verifying the NetBill protocol as a motivating and specified example in the proposed language using the MCMAS model checker along with NuSMV and CWB-NC as benchmarks.

1 Introduction

Several approaches have been put forward to specify interaction protocols that regulate and coordinate interactions among autonomous and heterogenous agents in multi-agent systems. Recently, some approaches have formalized these protocols in terms of creation and manipulation of commitments [7, 10, 13, 20, 21]. This kind of interaction protocols are called commitment protocols. Other approaches have been proposed to specify interaction protocols using computational logic-based languages [1, 3, 2] or a modified version of finite state machines that enables recombination and reusability of interaction protocols [16].

This paper concerns with defining a declarative specification of commitment protocols using a new language that extends CTL^* introduced in [9] with modality of commitments and actions on these commitments. We adopt the commitment protocols as they are increasingly used in different applications such as business processes [10, 20], artificial institutions [13] and web-based applications [19] during the past years. These protocols have social semantics in terms of the commitments that capture interactions among agents. In fact, commitments support flexible executions and provide declarative

representations of protocols by enabling the interacting agents to reason about their actions [21]. This flexibility is related to the fact of accommodating exceptions that arise at run time by offering more alternatives (or computations) to handle these exceptions [20]. For example, a commitment deadline may be renegotiated among participating agents, or the merchant may prefer to deliver goods before receiving the agreed amount of money. The protocols with fewer alternatives are less flexible, which restrict the autonomy of the participants. Commitments also capture the intrinsic business meanings of the exchanged messages [10, 20] and provide a principled basis for checking compliance of agents with given protocols via capturing the interactions states [7]. As a result, there is a tradeoff between protocol flexibility and the complexity of verifying the compliance. In addition, specifying and designing the commitment protocols that ensure flexible interactions are necessary, but not sufficient to automatically verify the conformance of protocols with some desirable properties that meet the important requirements of multi-agent business processes. This is because the automatic verification of protocol specifications at design time (i.e., before the actual implementation) leads to reduce the cost of development process and increase confidence on the safety, efficiency and robustness.

The aim of this paper is to address the above challenges by formally specifying commitment protocols and verifying them against some given properties using symbolic model checking. In fact, this work is a continuation of our previous publication [12], which is mainly focused on developing a new logical model unifying the full semantics of commitment operations and semantics of social commitments within the same framework. Figure 1 gives an overview of our approach. We begin with develop-

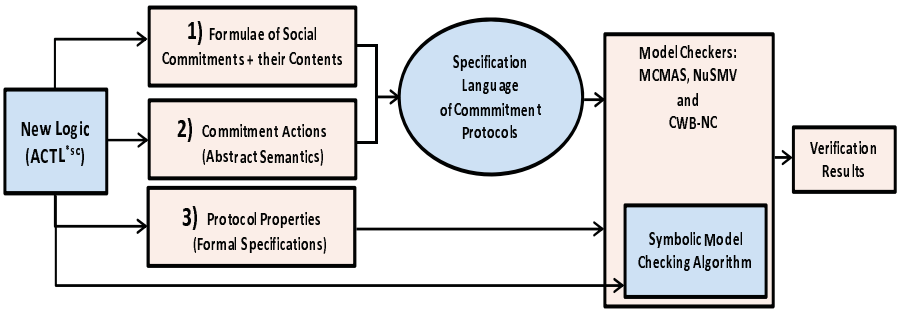

Fig. 1. A schematic view of our approach

ing a new language $\mathcal{L}$ by extending CTL^* with modality of social commitments and actions on these commitments. The resulting logic, called $ACTL^{*sc}$, is used to: (1) express well-formed formulae of commitments and their contents; (2) formally specify an abstract semantics of commitment actions that capture dynamic behavior of interacting agents; and (3) express protocol desirable properties to be checked such as fairness and liveness. By abstract semantics, we mean a semantics that does not define the meaning of all concrete action instances (e.g., withdraw and fulfill actions) but only the meaning of an abstract action denoted in this paper by θ. However, the concrete semantics of

commitments actions is given in our previous work [12]. On the other hand, the protocol properties are used to eliminate unwanted and bad agents' behaviors. Using social commitments and their actions, we define a new specification language of commitment protocols, which we use to regulate and coordinate the interaction among autonomous and heterogenous agents. We then proceed to develop a symbolic model checking algorithm for the proposed $ACTL^{*sc}$ logic based on OBDDs that support the compact representation of our approach. We present experimental results of verifying automatically the soundness of the NetBill protocol, as a running example, taken from e-business domain and specified using our specification language against some given properties. Finally, the implementation of this protocol is done using the MCMAS model checker [17] along with NuSMV [8] and CWB-NC [22] as benchmarks.

The remainder of this paper is organized as follows. Section 2 presents the $ACTL^{*sc}$: syntax and semantics. In Section 3, we use commitments and their actions to define a new specification language of commitment protocols. In Section 4, we encode the proposed logical model based on OBDDs and develop a symbolic model checking algorithm for this model. The implementation of the NetBill protocol and its verification using the MCMAS, NuSMV and CWB-NC model checkers with different experimental results is discussed in Section 5. The paper ends with some discussions of relevant literature in Section 6.

2 $ACTL^{*sc}$ Logic

In this section, we present $ACTL^{*sc}$ logic that we use to specify commitment protocols (see Sect.3) and express the properties to be verified (see Sect.5.2). We enhance CTL^* with social commitments and action formulae applied to these commitments. These modalities are needed for agent interactions and cannot be expressed using CTL^*. Formally, social commitments are related to the state of the world and denoted by $SC(Ag_1, Ag_2, \phi)$ where Ag_1 is the debtor, Ag_2 the creditor and ϕ a well-formed formula representing the commitment content. In some situations, especially in business scenarios, an agent wants to only commit about some facts when a certain condition is satisfied. We use conditional commitments to capture these situations [12]. Formally, conditional commitments are represented by $\tau \rightarrow SC(Ag_1, Ag_2, \phi)$ where "$\rightarrow$" is a logical implication, Ag_1, Ag_2 and ϕ have the above meanings and τ is a well-formed formula representing the commitment condition. We use $SC^c(Ag_1, Ag_2, \tau, \phi)$ as an abbreviation of $\tau \rightarrow SC(Ag_1, Ag_2, \phi)$. In this case, we have $SC(Ag_1, Ag_2, \phi) \triangleq SC^c(Ag_1, Ag_2, true, \phi)$.

The commitments can be manipulated or modified in a principled manner with the interaction progresses using commitment actions. These actions, reproduced from [18], are two and three-party actions. The former ones need only two agents to be performed such as: $Create$, $Withdraw$, $Fulfill$, $Violate$ and $Release$ actions. The latter ones need an intermediate agent to be completed such as: $Delegate$ and $Assign$ actions. Below the syntax and semantics of our language $\mathcal{L}$.

2.1 Syntax of $ACTL^{*sc}$

In the following, we use $\Phi_p = \{p, p_1, p_2, \ldots\}$ for a set of atomic propositions, $\Phi = \{\phi, \tau, \quad, \ldots\}$ for a set of propositional formulae, $\mathtt{AGT} = \{Ag, Ag_1, Ag_2, \ldots\}$ for a set

of agent names and $\texttt{ACT} = \{\theta, \theta_1, \theta_2, \ldots\}$ for a set of commitment actions. Agt and Θ are nonterminals corresponding to AGT and ACT respectively. Table 1 gives the formal syntax of the language $\mathcal{L}$ expressed in a BNF-like grammar where "::=" and "|" are meta-symbols of this grammar.

Table 1. The Syntax of $ACTL^{*sc}$-Logic

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

Formulae in $ACTL^{*sc}$ are classified into state formulae $\mathcal{S}$ and path formulae $\mathcal{P}$. The state formulae are formulae that hold on given states, while path formulae express temporal properties of paths and action formulae. The intuitive meanings of the most constructs of $ACTL^{*sc}$ are straightforward from CTL^* operators. The formula $A\phi$ (resp. $E\phi$) means that ϕ holds along all (some) paths starting at the current state. The formula $SC(Ag_1, Ag_2, \phi)$ means that agent Ag_1 commits towards agent Ag_2 that the path formula ϕ is true. Committing to path formulae is more expressive than committing to state formulae as state formulae are path formulae. The formula $X\phi$ means ϕ holds from the next state, $\phi_1 U \phi_2$ means ϕ_1 holds until ϕ_2 becomes true. The action formula $\theta(Ag_1, Ag_2, \mathcal{C})$ means that an action θ is performed by Ag_1 directed to Ag_2 on the commitment $\mathcal{C}$. For example, if θ is the *Assign* action, Ag_2 will be the agent to which the commitment is assigned. Furthermore, there are some useful abbreviations based on temporal operators: $F\phi \triangleq true\ U\phi$ (sometimes in the future) and $G\phi \triangleq \neg F \neg \phi$ (globally).

2.2 Semantics of $ACTL^{*sc}$

The semantics of this logic is interpreted with respect to the formal model M associated to the commitment protocol using a Kripke-structure as follows: $M = \langle \mathbb{S}, \texttt{ACT}, \texttt{AGT}, R_t, \mathbb{V}, \mathbb{R}_{sc}, \mathbb{L}, s_0 \rangle$ where: $\mathbb{S}$ is a set of states; ACT and AGT are defined above; $R_t \subseteq \mathbb{S} \times \texttt{AGT} \times \texttt{ACT} \times \mathbb{S}$ is a transition relation among states; $\mathbb{V} : \Phi_p \rightarrow 2^{\mathbb{S}}$ is an evaluation function; $\mathbb{R}_{sc} : \mathbb{S} \times \texttt{AGT} \times \texttt{AGT} \rightarrow 2^{\sigma}$, where σ is the set of all paths, is an accessibility modal relation that associates with a state s the set of possible paths along which the social commitments made by the debtor towards the creditor at s hold; $\mathbb{L} : \mathbb{S} \rightarrow 2^{\texttt{AGT} \times \texttt{AGT}}$ associates a given state s with a set of pairs and each pair represents the two interacting agents in s; and $s_0 \in \mathbb{S}$ is the initial state.

Instead of $(s_i, Ag_k, \theta_l, s_{i+1})$, transitions will be written as $s_i \xrightarrow{Ag_k:\theta_i} s_{i+1}$. The paths that path formulae are interpreted over have the form $P_i = s_i \xrightarrow{Ag_k:\theta_l} s_{i+1} \xrightarrow{Ag_{k+1}:\theta_{l+1}} s_{i+2} \ldots$ where $i \geq 0$. The set of all paths starting at s_i is denoted by σ^{s_i} and $\langle s_i, P_i \rangle$ refers to the path P_i starting at s_i. Also, when a state s_j is a part of a path P_j, we write $s_j \in P_j$. Excluding commitment modality and action formulae, the semantics of $ACTL^{*sc}$ state formulae is as usual (semantics of CTL^*) and a path formula satisfies

a state formula if the initial state in the path does so. $M, \langle s_i \rangle \models \phi$ means "the model M satisfies the state formula ϕ at s_i" and $M, \langle s_i, P_i \rangle \models \phi$ means "the model M satisfies the path formula ϕ along the path P_i starting at s_i". A state formula $SC(Ag_1, Ag_2, \phi)$ is satisfied in the model M at s_i iff the the content ϕ is true in every accessible path P_i, to which Ag_1 is committed towards Ag_2, starting from this state using $\mathbb{R}_{sc}$. Formally:

$M, \langle s_i \rangle \models SC(Ag_1, Ag_2, \phi)$ iff $\forall P_i \in \sigma^{s_i} : P_i \in \mathbb{R}_{sc}(s_i, Ag_1, Ag_2)$
$\Rightarrow M, \langle s_i, P_i \rangle \models \phi$ where "$\Rightarrow$" stands for implies.

To make the semantics computationally grounded, which is important for model checking, the accessibility relation $\mathbb{R}_{sc}$, that extends the original Kripke-structure, should be given a concrete (computational) interpretation to be able to describe our model as a computer program. This paper adopts a simple solution saying that if a commitment made by Ag_1 towards Ag_2 is satisfied at state s_i, then there is a path starting at this state (i.e., a possible computation) along which the commitment holds. The intuitive interpretation is as follows: when an agent Ag_1 commits towards another agent Ag_2 about ϕ, this means that there is at least a possible computation starting at this state satisfying ϕ. Formally, we use the following semantic rule:

$M, \langle s_i \rangle \models SC(Ag_1, Ag_2, \phi) \Rightarrow (Ag_1, Ag_2) \in \mathbb{L}(s_i)$ and $M, \langle s_i \rangle \models E\phi$

A path P_i starting at s_i satisfies $Create(Ag_1, Ag_2, SC(Ag_1, Ag_2, \phi))$ in the model M iff Ag_1: $Create$ is the label of the first transition on this path and $SC(Ag_1, Ag_2, \phi)$ holds in the next state s_{i+1}. Formally:

$M, \langle s_i, P_i \rangle \models Create(Ag_1, Ag_2, SC(Ag_1, Ag_2, \phi))$ iff $s_i \xrightarrow{Ag_1:Create} s_{i+1} \in R_t$
and $M, \langle s_{i+1} \rangle \models SC(Ag_1, Ag_2, \phi)$

Because of space limits, we only capture the abstract semantics of action formulae. However, the following concrete instances of these actions ($Fulfill$, $Violate$, $Withdraw$, $Release$, $Assign$, $Delegate$) are used to specify commitment protocols (see Sect.3). As mentioned in the introduction, the concrete semantics of these actions is entirely defined in our previous work [12]. The abstract semantics means that a path P_i starting at s_i satisfies $\theta(Ag_1, Ag_2, SC(Ag_1, Ag_2, \phi))$ in the model M iff $Ag_1 : \theta$ is the label of the first transition on this path and the commitment has been created in the past. Formally:

$M, \langle s_i, P_i \rangle \models \theta(Ag_1, Ag_2, SC(Ag_1, Ag_2, \phi))$ iff $s_i \xrightarrow{Ag_1:\theta} s_{i+1} \in R_t$ and
$\exists j \leq i, M, \langle s_j \rangle \models SC(Ag_1, Ag_2, \phi)$

3 Commitment Protocols

In this section, we present a formal specification language of commitment protocols derived from our logical model M (see Sect.2.2). For the sake of clarity, we use the NetBill protocol to demonstrate this specification.

3.1 Protocol Specification

In this paper, we define the specification of commitment protocols as a set of commitments capturing the business interactions among the interacting agents (or roles) at

design time. In addition to what 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. Autonomous agents communicate by exchanging messages and we assume that this exchanging is reliable, which means messages do not get lost and the communication channel ordered preserving.

The protocol specification begins with the commitment COM, which is followed by a message MSG. This message MSG may be directly mapped into commitment actions (captured by Θ) or into inform action. Specifically, MSG could either be withdrawn, fulfilled, violated, released, assigned, delegated or informed message. 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 Ag_1 to inform the creditor Ag_2 that a domain proposition holds. It is not a commitment action, but indirectly affects commitments by causing transformation from SC^c to SC commitments. The domain proposition Dom-Pro identifies the set of propositions (e.g., $Price\ request$) related to the application domain of the protocol. The inform message allows each agent to resolve its commitment, for example the merchant agent can use it to send a $refund$ to the customer agent. Each domain application can be represented by a suitable ontology. The formal specification of the proposed protocol, that is compatible with the standard protocols in business processes, is defined using a BNF-like grammar with meta-symbols: "::=" and "|" for the choice and ";" for action sequence (see Table 2).

Table 2. The formal specification of commitment protocols

Protocol	::= COM ; MSG
COM	::= $SC^c(Ag_1, Ag_2, Prop, Prop)$ | $SC(Ag_1, Ag_2, Prop)$
Prop	::= A well-formed formula in our $\mathcal{L}$
MSG	::= $Withdraw(Ag_1, COM)$ | $Fulfill(Ag_1, COM)$
	| $Violate(Ag_1, COM)$ | $Release(Ag_2, COM)$
	| $[Assign(Ag_2, Ag_3, COM)$ | $Delegate(Ag_1, Ag_3, COM)]$
	; $Create(Ag_1, COM)$; MSG
	| $Inform(Ag_1, Ag_2,$ Dom-Pro) | Dom-Pro
Dom-Pro	::= Identify domain propositions

The protocol terminates when the interacting agents do not have commitments to each other. Furthermore, the above specification language of commitment protocols can either be used at run time to reason about the actions [20] or compiled into a finite state machine at design time. At run time, the agents can logically compute their transitions using some reasoning rules. These rules enable agents to choose appropriate actions from the current situation and they are useful for the verification process [7] in relatively small systems. However, these rules are not enough to verify the correctness of the protocols against some given properties when the system is large and complex. For the purposes of this paper, the protocol specification is compiled into a finite state machine at design time where the business meaning of a state is given by the commitments that

hold in this state and the business meaning of actions are given by the actions applied on commitments (see Fig.2). We use symbolic model checking in order to verify the correctness of the protocols specified in our specification language (see Sect.4).

3.2 A Running Example

Let us consider the NetBill protocol taken from e-business domain as a running example to clarify the specification of the commitment protocols. This protocol begins at s_0 with a customer (Cus) requesting a quote for some desired goods like software programs or journal articles. This request is followed by the merchant (Mer) reply with sending the quote as an offer, which means creating a commitment. The Cus agent could either reject this offer, which means releasing this offer and the protocol will end at the failure state s_9 (see Fig.2), or accept this offer, which means creating a commitment at s_3. The Cus's commitment means that he is willing to pay the agreed amount if the Mer agent delivers the requested goods. At this state, the Cus agent still has two possibilities: to withdraw his commitment or to delegate it to a financial company (say Bank: B) to pay the Mer agent on his behalf. The most important thing here, the B agent can delegate this commitment to another bank B_1, which delegates the commitment back to the B agent. The banks B and B_1 delegate the commitment back and forth infinitely and this is presented by a loop at s_{11}. In a sound protocol, this behavior should be avoided (in Sect.5.2, we will show how to verify this issue).

The Mer agent, before delivering the goods to the Cus agent, can withdraw his offer at s_{10} and immaturely moving to the failure state s_9 after refunding payment to the Cus agent. However, when the Cus agent pays for the requested goods and the Mer agent delivers them (within a specified time), then the Mer agent fulfills his commitment at s_5 and then moved to sending the receipt to the Cus agent. Conversely, the Cus agent can pay for the requested goods without being delivered by the Mer agent within a specified time. In this case, the Mer agent violates his commitment at s_8 and

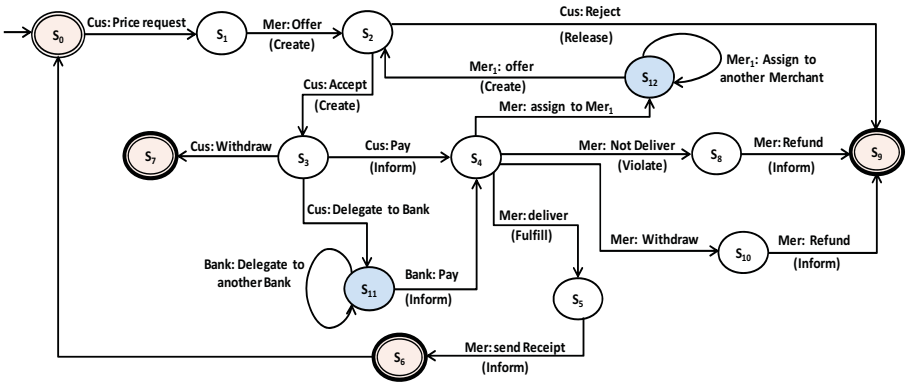

Fig. 2. Representation of NetBill protocol

immaturely moving to the failure state s_9 after refunding the payment to the Cus agent. Finally, the Mer agent, for some reasons, can assign his commitment to another mer-

Table 3. Business meaning of messages

Message	Meaning
Offer$(Mer, Cus, Pay, Deliver)$	Create$(Mer, SC^c(Mer, Cus, Pay, Deliver))$
Accept$(Cus, Mer, Good, Pay)$	Create$(Cus, SC^c(Cus, Mer, Good, Pay))$
Reject$(Cus, Mer, Good, Pay)$	Release$(Cus, SC^c(Mer, Cus, Good, Pay))$
Pay(Cus, Mer, Pay)	Inform(Cus, Mer, Pay)
Deliver$(Mer, Cus, Good)$	Fulfill$(Mer, SC(Mer, Cus, Good))$
Not Deliver$(Mer, Cus, Good)$	Violate$(Mer, SC(Mer, Cus, Good))$

chant (say Mer_1) at s_{12}. Specifically, the Mer agent releases the current commitment with the Cus agent and a new commitment between Cus and Mer_1 is created as a new offer to deliver the requested goods to the Cus agent. As for delegate scenario, the assign action can be repeated infinitely many times among interacting agents and this scenario, presented by a loop at s_{12}, is unwanted behavior in our protocol. Table 3 gives part of the NetBill protocol representation using our specification language along with the business meanings of the exchanged messages that are not expressed directly using commitment actions.

4 Symbolic Model Checking

Here, we describe how to encode the model M and the commitment protocol with Boolean variables and Boolean formulae. This encoding makes our representation more compact and enable us to use symbolic computations. Moreover, the verification algorithms that can operate on this representation are built progressively for symbolic model checking technique.

4.1 Boolean Encoding

In our approach, we use the standard procedure introduced in [9] to encode the concrete model $M = \langle \mathbb{S}, \mathtt{ACT}, \mathtt{AGT}, R_t, \mathbb{V}, \mathbb{L}, s_0 \rangle$ with OBDDs. The number of Boolean variables required to encode states $\mathbb{S}$ is $\mathcal{N} = \lceil \log_2 |\mathbb{S}| \rceil$ where $|\mathbb{S}|$ is the number of states in the model. Let $\overline{v} = \{v_1, \ldots, v_{\mathcal{N}}\}$ be a vector of $\mathcal{N}$ Boolean variables encoding each element $s \in \mathbb{S}$. Each tuple $\overline{v} = \{v_1, \ldots, v_{\mathcal{N}}\}$ is then identified with a Boolean formula, represented by a conjunction of variables or their negation. Thus, the set of states is encoded by taking the disjunction of the Boolean formulae encoding the states. We introduce $\mathcal{N}$ more variables to encode the "destination" state in a transition by means of vector $\overline{v'} = (v'_1, \ldots, v'_{\mathcal{N}})$, $\mathcal{O} = \lceil \log_2 |\mathtt{ACT}| \rceil$ and $\mathcal{A} = \lceil \log_2 |\mathtt{AGT}| \rceil$ variables to encode actions and agents resp. This representation allows us to encode the transition relations in R_t. Let us consider a generic pair $R_{t_1} = (s, Ag, \theta, s')$ be a transition relation in R_t, then its Boolean representation is given by $\overline{v} \wedge \overline{Ag} \wedge \overline{\theta} \wedge \overline{v'}$ in which $\overline{v}$ and $\overline{v'}$ are the Boolean representation of states s and s' respectively, $\overline{Ag}$ is the Boolean encoding for the agent and $\overline{\theta}$ is the Boolean encoding for the action. The Boolean formula

$f_{R_{t_1}}$ corresponding to R_{t_1} is obtained by taking the disjunction of all the possible such pairs. Thus, the Boolean formula f_{R_t} corresponding to the whole transition relation in our model is encoded by taking the conjunction of all the transition relations in R_t, where n is the number of transition relations.

$$f_{R_t}(v_1, \ldots, v_{\mathcal{N}}, Ag_1, \ldots, Ag_{\mathcal{A}}, \theta_1, \ldots, \theta_{\mathcal{O}}, v'_1, \ldots, v'_{\mathcal{N}}) = \bigwedge_{i=1}^{n} f_{R_{t_i}}$$

The evaluation function $\mathbb{V}$ is translated into a Boolean function $f_{\mathbb{V}} : \Phi_p \rightarrow B(v_1, \ldots, v_{\mathcal{N}})$ taking atomic proposition and producing the set $B(v_1, \ldots, v_{\mathcal{N}})$ of Boolean functions when a given atomic proposition is true. For example, given atomic proposition $p \in \Phi_p$, then $f_{\mathbb{V}}(p)$ is a Boolean function encoding the set of states where p is true. In the same way, the function $\mathbb{L}$ is translated into a Boolean function $f_{\mathbb{L}} : \mathbb{S} \rightarrow B(v_1, \ldots, v_{\mathcal{N}})$ taking a state and associating the set $B(v_1, \ldots, v_{\mathcal{N}})$ of Boolean functions representing the two interacting agents having a commitment made at this state. The Boolean encoding process is completed by encoding the initial state $s_0 \in \mathbb{S}$ as a set of Boolean variables like each member in $\mathbb{S}$.

Figure 3 depicts our verification workflow, which is performed in three phases. It starts with the specification of a commitment protocol as an input file written in the Interpreted Systems Programming Language (ISPL) for MCMAS, in the SMV language for NuSMV and in the Calculus of Communicating System (CCS) for CWB-NC. In the middle phase, the protocol properties to be checked are expressed in $ALTL^{sc}$ and $ACTL^{sc}$, which are Linear Temporal Logic (LTL), Computational Tree Logic (CTL) [9] augmented with commitments and their actions. In the last phase, the interpreted protocol specification and properties are the arguments of the model checking algorithm that computes the truth value of each property with respect to this specification.

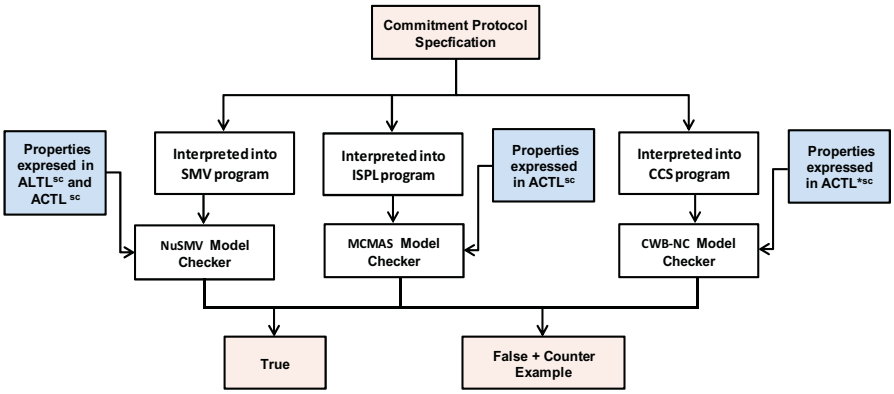

Fig. 3. Verification workflow of the protocol

4.2 Symbolic Model Checking Algorithm

In a nutshell, given the model M representing our protocol and a logical formula ϕ describing a property, the model checking is defined as the problem of computing whether

the model M satisfies ϕ (i.e. $M \models \phi$) or not (i.e. $M \not\models \phi$). Like proposed in [9] for CTL^* logic, in our approach the problem of model checking $ACTL^{*sc}$ formulae can be reduced to the problem of checking $ALTL^{sc}$ and $ACTL^{sc}$ formulae. Figure 4 depicts the expressive powers of the main components of our logic in which $ALTL^{sc}$

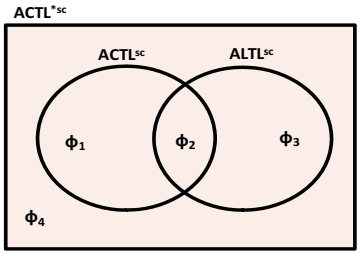

Fig. 4. The expressive powers of $ACTL^{*sc}$, $ACTL^{sc}$ and $ALTL^{sc}$

formulae (e.g., ϕ_3) are $ACTL^{*sc}$ path formulae in which the state sub-formulae are restricted to atomic propositions. Whilst, $ACTL^{sc}$ formulae (e.g., ϕ_1) are $ACTL^{*sc}$ formulae where every occurrence of a path operator is immediately preceded by a path quantifier. The formulae belonging to the intersection (e.g., ϕ_2) can be expressed in $ACTL^{sc}$ and $ALTL^{sc}$. However, the formulae outside the union (e.g., ϕ_4) can only be expressed in $ACTL^{*sc}$, which are usually defined as conjunctions or disjunctions of $ACTL^{sc}$ and $ALTL^{sc}$ formulae.

In our approach, for a given model M and for a given $ACTL^{*sc}$ formula ϕ, the algorithm $SMC(\phi, M)$ (see Table 4) computes the Boolean formula encoding the set of states where ϕ holds, we write this set as $[\![\phi]\!]$. Similarly to the standard OBDD-based model checking for CTL and LTL [9, 15], the Boolean formulae resulting from this algorithm can be manipulated using OBDDs. The OBDD for the set of reachable states in the model M is compared to OBDD corresponding to each formula. If the two are equivalent (i.e., the formula holds in the model), then the algorithm reports a positive output (or true), otherwise a negative output (or false) plus counter example (see Fig.3) is produced.

Table 4. $ACTL^*$ symbolic model checking algorithm

1. $SMC(\phi, M)$ {// for $ACTL^{*sc}$ formulae
2. ϕ is an atomic formula: return $\mathbb{V}(\phi)$;
3. ϕ is $\neg\phi_1$: return $\mathbb{S}\backslash SMC(\phi_1, M)$;
4. ϕ is $\phi_1 \vee \phi_2$: return $SMC(\phi_1, M) \cup SMC(\phi_2, M)$;
5. ϕ is $SC(Ag_1, Ag_2, \phi_1)$: return $SMC_{sc}(Ag_1, Ag_2, \phi_1, M)$;
6. ϕ is $SC^c(Ag_1, Ag_2, \tau_1, \phi_1)$: return $SMC_{scc}(Ag_1, Ag_2, \tau_1, \phi_1, M)$;
7. ϕ is $\theta(Ag_1, Ag_2, \mathcal{C})$: return $SMC_{act}(\theta, Ag_1, Ag_2, \mathcal{C}, M)$;
8. ϕ is $E\phi_1$: return $SMC_E(\phi_1, M)$;
9. }

When the formula ϕ is of the form $SC(Ag_1, Ag_2, \phi_1)$, then the algorithm calls the

Table 5. The procedure for checking $\phi = SC(Ag_1, Ag_1, \phi_1)$

$SMC_{sc}(Ag_1, Ag_2, \phi_1, M)$ {//for social commitment modality
$X = SMC_E(E\phi_1, M)$;
$Y = \{s \in \mathbb{S} \mid (Ag_1, Ag_2) \in \mathbb{L}(s)\}$;
return $X \cap Y$;
}

procedure $SMC_{sc}(Ag_1, Ag_2, \phi_1, M)$, which begins with computing the set of states where the existential path formula ϕ_1 holds (i.e., $[\![E\phi_1]\!]$) using the standard procedure $SMC_E(E\phi_1, M)$ (see Table 5). Then builds the set of states in which the agent Ag_1 commits towards the agent Ag_2 to bring about ϕ with respect to the function $\mathbb{L}$. The set of states satisfying $SC(Ag_1, Ag_2, \phi_1)$ is finally computed by taking the conjunction of

Table 6. The procedure for checking $\phi = SC^c(Ag_1, Ag_1, \tau_1, \phi_1)$

$SMC_{scc}(Ag_1, Ag_2, \tau_1, \phi_1, M)$ {//for a conditional commitment modality
$X = SMC(\tau_1, M)$;
return $\neg X \cap SMC_{sc}(Ag_1, Ag_2, \phi_1, M)$;
}

the two sets. The procedures $SMC_{scc}(Ag_1, Ag_2, \tau_1, \phi_1, M)$ and $SMC_{act}(\theta, Ag_1, Ag_2, \mathcal{C}, M)$ for the formulae of the form $SC^c(Ag_1, Ag_2, \tau_1, \phi_1)$ and $\theta(Ag_1, Ag_2, \mathcal{C})$ are presented in Tables 6 and 7 respectively.

Table 7. The procedure for checking $\phi = \theta(Ag_1, Ag_2, \mathcal{C})$

$SMC_{act}(\theta, Ag_1, Ag_2, \mathcal{C}, M)$ {//for action formulae
$X = \{s \mid \exists s' \in \mathbb{S}$ and $f_{R_t}(s, Ag, \theta, s')\}$;
$Y = SMC_{sc}(Ag_1, Ag_2, \phi_1, M)$;
$Z = \{s \mid \exists s' \in X \; and \; \exists P \in \sigma^s : s' \in P\}$;
return $Y \cap Z$;
}

The procedure for computing the set of states satisfying $ACTL^{*sc}$ formula ϕ of the form $E\phi_1$ is shown in Table 8. This procedure $SMC_E(\phi_1, M)$ first checks if the formula ϕ_1 is $ACTL^{sc}$ formula, then it calls the model checking algorithm $SMC_{actl}(\phi_2, M)$ for $ACTL^{sc}$ to compute the set of states satisfies this formula. Otherwise, it calls the model checking algorithm $SMC_{altl}(\phi', M)$ for $ALTL^{sc}$ after replacing each maximal state sub-formula with a new atomic proposition and the evaluation function is adjusted by adding the set of states that satisfy the new proposition to the existing states (for more details see [9]). That is, if $E_1, \ldots, E_k$ are the maximal state sub-

formulae of ϕ' (i.e., for all $E\ \ _i$ in ϕ such that $E\ \ _i$ is not contained in any other maximal state sub-formula of ϕ), $p_1, \ldots, p_k$ are atomic propositions, then the formula ϕ' is obtained by replacing each sub-formula $E\ \ _i$ with atomic proposition p_i. The resulting formula ϕ' is a pure $ALTL^{sc}$ path formula (to clarify this notion see Example1). Like the standard algorithm of CTL (resp. LTL) formulae [15], the $SMC_{actl}(\phi_2, M)$ (resp. $SMC_{altl}(\phi', M)$) algorithm computes the set of states in which the formula ϕ_2 (resp. ϕ') holds. Moreover, $SMC_{actl,EX}$, $SMC_{actl,EG}$, $SMC_{actl,EU}$, $SMC_{altl,X}$ and $SMC_{altl,U}$ are the standard procedures defined in [15, 9] to compute EX, EG and EU of $ACTL^{sc}$ operators and X and U of $ALTL^{sc}$ operators resp.

Table 8. The procedure for checking $\phi = E\phi_1$

$SMC_E(\phi_1, M)\{$ // for existential formula
If ϕ_1 is an $ACTL^{sc}$ formula: return $SMC_{actl}(\phi_2, M)$;
Otherwise $\phi' = \phi_1[p_1/E\ \ _1, \ldots, p_1/\ \ _k]$;
for all $E\ \ _i \in \phi_1$;
For i:=1,..., k do
if $s \in \mathbb{V}(E\ \ _i)\ then\ \mathbb{V}(E\ \ _i) := \mathbb{V}(E\ \ _i) \cup \mathbb{V}(p_i)$;
$\Phi_p := \Phi_p \cup p_i$;
end for all;
return $SMC_{altl}(\phi', M)$;
$\}$
$SMC_{actl}(\phi_2, M)$ {// for $ACTL^{sc}$ formula
ϕ_2 is an atomic formula: return $\mathbb{V}(\phi_2)$;
ϕ_2 is $\neg\phi'$: return $\mathbb{S}\backslash SMC_{actl}(\phi', M)$;
ϕ_2 is $\phi' \vee \phi''$: return $SMC_{actl}(\phi', M') \cup SMC_{actl}(\phi'', M)$;
ϕ_2 is $EX\phi'$: return $SMC_{actl,EX}(\phi', M)$;
ϕ_2 is $EG\phi'$: return $SMC_{actl,EG}(\phi', M)$;
ϕ_2 is $E[\phi' \cup \phi'']$: return $SMC_{actl,EU}(\phi', \phi'', M)$;
$\}$
$SMC_{altl}(\phi', M)$ { // for $ALTL^{sc}$ formula
ϕ' is an atomic formula: return $\mathbb{V}(\phi')$;
ϕ' is $\neg\phi_1$: return $\mathbb{S}\backslash SMC_{altl}(\phi_1, M)$;
ϕ' is $\phi_1 \vee \phi_2$: return $SMC_{altl}(\phi_1, M) \cup SMC_{altl}(\phi_2, M)$;
ϕ' is $X\phi_1$: return $SMC_{altl,X}(\phi_1, M)$;
ϕ' is $\phi_1 \cup \phi_2$: return $SMC_{altl,U}(\phi_1, \phi_2, M)$;
$\}$

The time complexity of the proposed algorithm depends on the time complexity of the model checking algorithms for $ACTL^{sc}$ and $ALTL^{sc}$. The complexity of $ACTL^{sc}$ model checking problem (like CTL) is P-complete with respect to the size of an explicit model and PSPACE-complete in case of $ALTL^{sc}$ (like LTL). As a result, the time complexity of $ACTL^{*sc}$ model checking problem in the worst case is PSPACE-complete with respect to symbolic representations.

Example 1. Let ϕ be an $ACTL^{*sc}$ formula, which means that whenever the Cus agent does not pay for the goods, then either the goods will be never delivered or the Cus

agent will eventually withdrawn. Formally:
$\phi = AG(\neg Pay(Cus) \rightarrow A(G\neg Deliver(Mer) \vee FWithdraw(Cus)))$. In order to simplify the model checking, we consider only the existential path quantifier. Thus, ϕ is rewritten as: $\neg EF(\neg Pay(Cuse) \wedge E(F\ Deliver(Mer) \wedge G\ Withdraw))$

- At level 0: all atomic propositions: Pay(Cus), Deliver(Mer) and Withdraw(Cus) are checked.
- At level 1: the formula $\neg Pay(Cus)$ is checked to compute the set of states that satisfy it. The other formula of level 1, $E(F\ Deliver(Mer) \wedge G\ Withdraw(Cus))$, is a pure $ALTL^{sc}$ formula, which is checked by calling SMC_{altl} procedure.
- At level 2: the formula $(\neg Pay(Cus) \wedge E(F\ Deliver(Mer) \wedge G\ Withdraw(Cus)))$ is also checked by calling SMC_{altl} procedure as it is a pure $ALTL^{sc}$ formula.
- At level 3: the formula $E(F\ Deliver(Mer) \wedge G\ Withdraw(Cus))$ is first replaced by the atomic proposition p. The $ALTL^{sc}$ model checking algorithm is then applied to the pure $ALTL^{sc}$ formula $EF(\neg Pay(Cus) \wedge p)$. Finally, all states in the NetBill protocol satisfy the formula: $\neg EF(\neg Pay(Cus) \wedge p)$, which means that this property always holds for this protocol.

5 Implementation

Currently, there are many model checkers developed for different purposes. In this paper, we use MCMAS, a symbolic model checker [17] based on OBDDs to verify the proposed protocol against some properties. More specifically, MCMAS has been implemented in C^{++} and developed for verifying multi-agent systems. It is mainly used to check a variety of properties specified as CTL or $ACTL^{sc}$ formulae in our language. In MCMAS, the multi-agent systems are described by the ISPL language where the system is distinguished into two kinds of agents: environment agent and a set of standard agents. Environment agent is used to describe the boundary conditions and observations shared by standard agents and is modeled as a standard agent. Standard agent can be seen as an non-deterministic automaton with the following components: a set of local states (some of which are initial states), a set of actions, protocol functions and evolution functions that describe how the local states of the agents evolve based on their current local states and other agents' actions.

As benchmarks, we use NuSMV, a symbolic model checker [8] and CWB-NC, a non-symbolic model checker (or automata-based model checker). More specifically, NuSMV has been successfully adopted to model checking multi-agent systems. 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, the different components and functionalities of the system are described by finite state machines and translated into isolated and separated modules. These modules can be composed synchronously and asynchronously. This paper specifically uses NuSMV to check the properties expressed in $ALTL^{sc}$, which cannot be verified using MCMAS. Meanwhile, it also uses NuSMV to compare the verification results of checking $ACTL^{sc}$ properties obtained by MCMAS (see Sect.5.3).

On the other hand, CWB-NC uses Milner's Calculus of Communicating Systems (CCS) as the design language to model concurrent systems. In fact, CCS is a process

algebra language, which is a prototype specification language for reactive systems. CCS can be used not only to describe implementations of processes, but also the specifications of their expected behaviors. We elect CWB-NC because it uses $GCTL^*$ that generalizes CTL^* with actions and closes to our $ACTL^{*sc}$ formulae without considering social commitments. Thus, we adapt CWB-NC to capture commitment formulae and in this case CWB-NC takes as input the CCS code and $ACTL^{*sc}$ property and automatically checks the validity of this property by building its Alternating Büchi Tableau Automata (ABTA).

5.1 Translating Commitment Protocols

The main step in our verification workflow (see Fig.3) is to translate protocol specification into ISPL, SMV and CCS. In MCMAS, this process begins with translating a set of interacting agents (the Mer and Cus agents in our running example in Sect.3.2) into standard agents in Agent section and commitments into local variables in Vars section. Such variables are of enumeration type including all possible commitment states (see Fig.2), which verify whether the protocol is in a conformant state or not. The actions on commitments are directly expressed in Actions statement where such actions work as constraints to trigger or stop transitions among states. The translation is completed by declaring a set of initial states in InitStates section from which the protocol verification starts to compute the truth value of formulae that are declared in Formulae section.

On the other hand, in SMV, the set of interacting agents are translated into isolated modules, which are instantiated in the main module that also includes the definition of the initial conditions using the INIT statement and the formulae that need to be checked using the SPEC statement. The commitment states are defined in SMV variables in VAR statement. Such states with actions are used as reasoning rules to evolve the state changes. The transition relation between commitment states and their actions is described using TRANS statement where all necessary transitions are defined (see Fig.2). The TRANS statement proceeds with defining the local initial condition using the INIT statement and includes the definition of the evolution function that mainely captures the transition relations using the next statement and Case statement that represents agent's choices. Finally, in CCS, each agent in our protocol is represented by a set of processes and each process is specified by proc statement. The states of a commitment are captured by the set of actions performed on this commitment. Each transition relation is represented by the action labeling this transition followed by another process. For example, let M0,M1,M2 be three processes:

```
proc M0 = 'Request(Cus).M1
proc M1 = Accept(Mer).M2 + Release(Mer).M0
```

means after receiving a request from the Cus agent, the Mer agent accepts or releases. The formulae that we want to check are written in a special file with extension .gctl using $ACTL^{*sc}$.

5.2 Verifying Commitment Protocols

To automatically verify the soundness of the proposed protocol specifications, we need to introduce some desirable properties. In fact, some proposals have been put forward

to classify these properties that satisfy different requirements of the commitment protocols [21, 11]. Specifically, P. Yolum has verified the correctness of the commitment protocols at design time with respect to three kinds of a generic properties: effectiveness, consistency and robustness [21]. N. Desai et al. have classified these properties into two classes: general properties and protocol-specific properties to verify the commitment protocols and their composition [11]. In the following, we specify some temporal properties: fairness, safety, liveness, reachability and deadlock-freedom using $ALTL^{sc}$, $ACTL^{sc}$ and $ACTL^{*sc}$. These properties are more general than the properties introduced in [11] and satisfy the same functionalities of the properties presented in [21]. The differences and similarities of our properties with the properties developed in [11, 21] are explained in Section 6.

1. Fairness constraint: it is needed to rule out unwanted behaviors of agents (e.g. a printer being locked forever by a single agent) [9]. In our protocol, if we define the formula: $AG(AF\ \neg Delegate(Bank))$ as a fairness constraint, then a computation path is fair iff infinitely often the $Bank$ agent does not delegate commitments. This constraint will enable us to avoid situations such as the banks agents delegate the commitment back and forth infinitely (see Sect.3.2). Thus, by considering fairness constraints, the protocol's fairness paths include only the paths that the interacting agents can follow to satisfy their desired states fairly.
2. Safety: means that "something bad never happens". This property is generally expressed by $AG\neg p$ where p characterizes a "bad" situation, which should be avoided. For example, in our protocol a bad situation is: in all paths the Cus agent always pays the agreed payment, but the Mer agent will not eventually deliver the requested goods in all paths starting from the state where the Cus agent has payed: $AG(\neg(Pay(Cus)\ \wedge AF(AG\neg Deliver(Mer)))$.
3. Liveness: means that "something good will eventually happen". For example, in all paths where the Cus agent eventually pays the agreed payment, then there is a path in its future the Mer agent will either (1) deliver the goods and in all paths in the future he will send the receipt; (2) withdraw the commitment; or (3) violate it and in all paths in the future he will refund the payment to the Cus agent: $AF(Pay(Cus) \rightarrow EF((Deliver(Mer) \wedge AFReceipt(Mer)) \vee (Withdraw(Mer) \wedge AFRefund(Mer)) \bar{\vee} (Not\ Deliver\ (Mer) \wedge AFRefund\ (Mer))))$. Another example of liveness is: in all paths where the Mer agent is eventually in "withdraw" state or "not deliver" state, then he will eventually, in all paths starting at these states, refund the payment to the Cus agent: $AF(Withdraw(Mer) \vee Not\ Deliver(Mer) \rightarrow AFRefund(Mer))$.
4. Reachability: a particular situation can be reached from the initial state via some computation sequences. For example, in all paths in the future, there is a possibility for the Mer agent to deliver the request goods to the Cus agent: $AF(EFDeliver(Mer))$. Also, this property could be used to show the absence of deadlock in our protocol. Formally: $\neg AF(EFDeliver(Mer))$, which means that the deadlock is the negation of the reachability property, which is supposed to be false.

The above formulae are only some examples, which were given for the fragment of $ACTL^{sc}$. By considering the definition of LTL given in [9], it is clear that the fairness

and safety properties have equivalent $ALTL^{sc}$ formulae. Furthermore, the first example of liveness property cannot be expressed in $ALTL^{sc}$ because of the existence quantifier and the second example of this property also cannot be expressed in $ALTL^{sc}$ because the candidate $ALTL^{sc}$ formula: $AF(\neg(Withdraw(Mer) \vee Not\ Deliver(Mer)) \wedge FRefund(Mer))$ is weaker than the $ACTL^{sc}$ formula. Note that this does not mean that a meaningful formula in $ACTL^{sc}$ does not have an equivalent formula in $ALTL^{sc}$, for example $AG(AF\neg Delegate(Bank))$ has equivalent $ALTL^{sc}$ formula: $AGF\neg Delegate(Bank)$. In fact, when considering conjunctions or disjunctions of the above formulae, we can construct a formula that is $ACTL^{*sc}$ formula, but neither $ACTL^{sc}$ nor $ALTL^{sc}$.

5.3 Experimental Results

In this section, we present three experimental results using the MCMAS, NuSMV and CWB-NC model checkers. From business process point of view, these experiments try to capture some real-life business scenarios that our specification language of commitment protocols can effectively formalize. In the first experiment we only consider the simple business scenario between two agents (*Cus* and *Mer*) that use the NetBill protocol to coordinate their interactions starting when the Cus agent requests some goods until the Mer agent delivers them. In the second one, we extend the first experiment by adding an agent (say *Mer1*) to which a commitment can be assigned and in the third one, we give the Cus agent the possibility to delegate his commitment to another agent (say *Bank*) to complete the commitment on his behalf.

These experiments show that the current widely known symbolic model checkers are supporting the verification of $ACTL^{sc}$. We use MCMAS as it includes an agent-based specification and is easy to use. However, for $ALTL^{sc}$ formulae that cannot be checked with MCMAS, we use NuSMV. On the other hand, we use CWB-NC in order to verify $ACTL^{*sc}$ formulae as it is the only model checker that can accomplish this kind of verification. Moreover, the use of CWB-NC is motivated by the fact that we need to compare the verification results obtained by the symbolic approach with the automata-based technique to demonstrate the main benefits of our approach.

Results Analysis:
We start our analysis with defining the main criteria that we used to evaluate the performance of model checking theoretically and practically. These criteria are generally related to the model size and total time (i.e., the time of building the model and verification time). Theoretically, we define the size of the model as $|M| = |\mathbb{S}| + |R_t|$, where $|\mathbb{S}|$ is the state space and $|R_t|$ is the relation space. For example, in the third experiment we have: $|\mathbb{S}| = |\mathbb{S}_{Cus}| \times |\mathbb{S}_{Mer}| \times |\mathbb{S}_{Mer_1}| \times |\mathbb{S}_{Bank}| \times |\mathbb{S}_{CP}|$, where $|\mathbb{S}_{Ag_i}|$ is the number of states for $Ag_i \in \{Cus, Mer, Mer_1, Bank\}$ and $|\mathbb{S}_{CP}|$ is the number of states of the commitment protocol. An agent state is described in terms of the possible messages, which he uses to interact with the other and each message is described by a set of states. For example, the two-party actions need 2 states and three-party actions need 3 states. Thus, for the Cus, Mer, Mer_1 and $Bank$ agents in the third experiment, we have 16, 20, 10, 6 states respectively. The protocol is described by the legal actions (see Fig.2), so it needs 13 states. In total, the number of states needed for this

experiment is $|\mathbb{S}| = 249600 \approx 2.5.\ 10^5$. To calculate $|R_t|$, we have to consider the operators of $ACTL^{*sc}$, where the total number is 10 operators. We can then approximate $|R_t|$ by $10.|\mathbb{S}|^2$. So we have $|M| \approx 10.|\mathbb{S}|^2 \approx 6.23.\ 10^{11}$. The model size in our three experiments are shown in Table 9.

Table 9. The model size in the three experiments

	Exp.1	Exp.2	Exp.3
The theoretical model size	$\approx 3.5.\ 10^7$	$\approx 9.73.\ 10^{10}$	$\approx 6.23.\ 10^{11}$

Hereafter, we only analyze the practical results of verifying the above properties of the third experiment because it considers a rich variety of interacting agents. Table 10 displays the full results of these three experiments and the execution times (in seconds) on a laptop running Windows Vista Business on Intel Core 2 Duo CPU T5450 1.66 GHz with 2.00GB memory. The total time for verifying our protocol using MCMAS increases when augmenting the number of agents from 2 to 4 agents. This is normally because the number of OBDD variables needed to encode agents states increases with the number of interacting agents.

As we mentioned, the NuSMV and CWB-NC model checkers are used as benchmarks to evaluate the results obtained by MCMAS. The three experimental results using the same machine as for MCMAS are given in Table 10. In terms of symbolic model checkers, these results reveal that the number of OBDD variables (which reflect the model size) is greater in NuSMV than in MCMAS, but the total time in NuSMV is better than in MCMAS as some optimization techniques are implemented in NuSMV such as on-fly-model checking and cashing. In terms of non-symbolic model checkers, as we expect, when the size model is small, then the total time in CWB-NC is better than in MCMAS and NuSMV (see Exp.1 and Exp.2 in Table 10). However, when the state space increases (as in Exp.3), then the total time in MCMAS and NuSMV is going to be better than in CWB-NC. Note that, we put "–" in Table 10 because CWB-NC does not use OBDD variables.

To conclude this section, the MCMAS model checker underpins different logics such as CTL-logic, supports agents' specifications and performs moderately better than both NuSMV (in terms of OBDD variables) and CWB-NC (in terms of the total time).

Table 10. The statistical 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
OBDD Variables	24	39	49	33	53	67	–	–	–
Number of Agents	2	3	4	2	3	4	2	3	4
Total Time(sec)	≈ 0.52	≈ 2	≈ 6	≈ 0.23	≈ 1.11	≈ 1.98	≈ 0.094	≈ 0.564	≈ 8.814

6 Related Work

We review the recent literature with respect to our work. In terms of defining commitment protocols, Chopra et al. [7] have defined the commitment protocol as a transition system and investigated an agent's compliance with the protocol and interoperability with other agents by considering only the two-part actions. Desai et al. [10] have used the Web Ontology Language (WOL) to specify the commitment protocols and describe some concepts of composing multiple protocol specifications to simplify the development of business processes. Fornara et al. [13] have proposed an application independent method to define interaction protocols having social semantics in artificial institutions. They treat commitments like objects as in object-oriented programming and do not consider delegation, assignment, or release of commitment. Yolum and Singh [20] have used commitment operations to show how to build and execute commitment protocols and how to reason about them 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 using a more compatible logic with agent open choices, which is an extension of CTL^*-logic. We have also developed symbolic model checking algorithm, which can verify the proposed modality and action formulae without suffering from the state explosion problem that could be occurred in large systems as in [4].

In terms of verifying the conformance of commitment protocols, Venkatraman et al. [19] have presented an approach for testing whether the behavior of an agent in open systems complies with a commitment protocol specified in CTL-logic. The proposed approach complements this work by introducing the model checking technique and the verification of the structural properties geared toward the interactions between agents. Desai et al., [11] has introduced model checking using Promela and Spin to verify commitment-based business protocols and their compositions based on LTL logic. The authors also define general properties in terms of the capabilities of Spin model checker to verify the deadlocks and livelocks, where deadlock can result from the contradiction among composition axioms without considering fairness constraints and reachability properties. They introduced some "protocol-specific properties", which can be defined using the safety property. Moreover, the specification language here is not only $ALTL^{sc}$ specification, but also $ACTL^{sc}$ specification. We also use MCMAS and NuSMV tools, which underpin symbolic representation based on OBDDs that is computationally more efficient than automata-based model checkers such as Spin. Baldoni et al. [3] have addressed the problem of verifying that a given protocol implementation using a logical language conforms to its AUML specification. Alberti et al. [1] have considered the problem of verifying on the fly the compliance of the agents' behaviors to protocols specified using a logic-based framework. These approaches are different from the technique presented in this paper in the sense that they are not based on model checking and do not address the problem of verifying if a protocol satisfies some given properties. Aldewereld et al. [2] have used a theorem proving method to verify the norm compliance of interaction protocols. This norm (e.g., permission) and some temporal properties (e.g. safety and liveness) that need to be checked are expressed in LTL, which is different from our approach that uses symbolic model checking and social commitments.

Recently, Leôn-Soto [16] has presented a model based on the "state-action" space to develop interaction protocols from a global perspective with the flexibility to recombine and reuse them in different scenarios. However, these approaches do not consider the formal specification of the interactions protocols as well as the automatic verification of these protocols. Giordano and her colleagues [14] addressed the problem of specifying and verifying systems of communicating agents in a dynamic linear time temporal logic (DLTL). However, the dynamic aspect of our logic is represented by action formulae and not by strengthening the until operator by indexing it with the regular programs of dynamic logic. In [4], the interacting agent-based systems communicate by combing and reasoning about dialogue games, the verification method is based on the translation of formula into a variant of alternating tree automata called alternating Büchi tableau automata. Unlike this approach, our verification algorithm is encoded using OBDDs. Thereby, it avoids building or exploring the state space corresponding to the model explicitly. As a result, it is more suitable for complex and large systems.

In terms of commitment protocol properties, Yolum in [21] 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. The proposed properties meet these requirements in the sense that the reachability and deadlock-freedom 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 accompanied with recover states that capture the protocol failures such as if the Mer agent withdraws or violates his commitment, then he must refund the payment to the Cus agent. Hence, our approach can be applied to verify the protocol's properties defined in [21]. As a future work, we plan to expand the formalization of commitment protocol with metacommitments and apply our symbolic model checking to verify the business interactions between agent-based web services.

Acknowledgements

We would like to thank the reviewers for their valuable comments and suggestions. Jamal Bentahar 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. Alberti, M., Daolio, D., Torroni, P., Gavanelli, M., Lamma, E., Mello, P.: Specification and verification of agent interaction protocols in a logic-based system. In: Proc. of ACM Symposium on Applied Computing, pp. 72–78. ACM Press (2004)
2. Aldewereld, H., Vázquez-Salceda, J., Dignum, F., Meyer, J.-J. Ch.: Verifying norm compliancy of protocols. In: Bossier, O., Padget, J., Dignum, V., Lindeman, G., Matson, E., Ossowski, S., Sichman, J., Vázquez-Salceda, J. (eds.), Coordination, Organisation, Institutions and Norms in Agent Systems, vol.3913 of LNAI, pp. 231–245. Springer (2006)
3. Baldoni, M., Baroglio, C., Martelli, A., Patti, V., Schifanella, C.: Verifying protocol conformance for logic-based communicating agents. In: Lenite, J.A., Torroni, P. (eds.), Computational Logic in Multi-Agent Systems (CLIMA V), vol.3487 of LNAI, pp. 196–212. Springer (2005)

4. Bentahar, J., Meyer, J.-J. Ch., Wan, W.: Model checking communicative agent-based systems. Knowledge Based Systems, 22(3): 142–159 (2009)
5. Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking agent-speak. In: Proc. of the 2nd International Joint Conference on AAMAS, pp. 409–416. ACM Press (2003)
6. Chopra, A.K., Singh, M.P.: Multi-agent commitment alignment. In: Proc. of the 8th International Conference on AAMAS, vol.2, pp. 937–944. ACM Press (2009)
7. Chopra, A.K., Singh, M.P.: Producing compliant interactions: Conformance, coverage and interoperability. In: DALT, vol.4324 of LNCS, pp. 1–15. Springer (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: Proc. of the 14th International Conference on CAV, vol.2404 of LNCS, pp. 359–364. Springer (2002)
9. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press, Cambridge (1999)
10. Desai, N., Mallya, A.U., Chopra, A.K., Singh, M.P.: Interaction protocols as design abstractions for business processes. IEEE Transactions on Software Engineering, 31(12): 1015–1027 (2005)
11. Desai, N., Cheng, Z., Chopra, A.K., Singh, M.P.: Toward verification of commitment protocols and their compositions. In: Proc. of the 6th International Joint Conference on AAMAS, pp. 144–146. ACM Press (2007)
12. El-Menshawy, M., Bentahar, J., Dssouli, R.: Verifiable semantic model for agent interactions using social commitments. In: Dastani, M., El Fallah, A.S., Leite, J.A., Torroni, P. (eds.), Languages, Methodologies and Development tools for Multi-agent Systems, LNAI. Springer (2009) (In Press)
13. Fornara, N., Colombetti, M.: Specifying artificial institutions in the event calculus (Chap.14). In: Dignum, V. (editor), Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models, IGI Global, pp. 335–366 (2009)
14. Giordano, L., Martelli, A., Schwind, C.: Verifying communicating agents by model checking in a temporal action logic. In: Logics in Artificial Intelligence (JELIA), vol.3229 of LNAI, pp. 57–69. Springer (2004)
15. Huth, M., Ryan, M.: Logic in computer science: Modelling and reasoning about systems (2nd edition). Cambridge University Press (2004)
16. Leôn-Soto, E.: Modeling interaction protocols as modular and reusable 1st class objects. In: Proc. of Agent-Based Technologies and Applications for Enterprise Interoperability, vol.25 of LNBIP, pp. 174–196 (2009)
17. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A model checker for the verification of multi-agent systems. In: CAV, pp. 682–688 (2009)
18. Singh, M.P.: An ontology for commitments in multi-agent systems: Toward a unification of normative concepts. In: AI and Law, vol.7, pp. 97–113 (1999)
19. Venkatraman, M., Singh, M.P.: Verifying compliance with commitment protocols: Enabling open web-based multi-agent systems. In: Autonomous Agents and Multi-Agent Systems, pp. 217–236 (1999)
20. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: Applying event calculus planning using commitments. In: Proc. of the 1st International Joint Conference on AAMAS, pp. 527–534. ACM Press (2002)
21. Yolum, P.: Design time analysis of multi-agent protocols. Data Knowladge Engneering, 63(1): 137–154. Elsevier (2007)
22. Zhang, D., Cleaveland, R., Stark, E.W.: The integrated CWB-NC/PIOA tool for functional verification and performance analysis of concurrent systems. In: TACAS, vol.2619 of LNCS, pp. 431–436. Springer (2003)