Elsevier Editorial System(tm) for Expert Systems With Applications
Manuscript Draft

Manuscript Number: ESWA-D-12-01475

Title: Verifying Conformance of Multi-Agent Commitment-based Protocols

Article Type: Full Length Article

Keywords: Multi-Agent Systems; Commitment-based Protocols; Reduction;
Symbolic Model Checking; Verification

Corresponding Author: Dr. Jamal Bentahar, Ph.D.

Corresponding Author's Institution: Concordia University

First Author: Mohamed El-Menshawy Mohamed, Ph.D.

Order of Authors: Mohamed El-Menshawy Mohamed, Ph.D.; Jamal Bentahar, Ph.D.; Warda El Kholy, Ph.D.; Rachida Dssouli, Ph.D.

Verifying Conformance of Multi-Agent Commitment-based Protocols

Mohamed El-Menshawy[a], Jamal Bentahar[a], Warda El Kholy[a], Rachida Dssouli[a]

[a] *Concordia University, Faculty of Engineering and Computer Science, Canada*

Abstract

Although several approaches have been proposed to specify multi-agent commitment-based protocols that capture flexible and rich interactions among autonomous and heterogeneous agents, very few of them synthesize their formal specification and automatic verification in an integrated framework. In this paper, we present a new logic-based language to specify commitment-based protocols, which is derived from ACTL^{*c}, a logic extending CTL* with modalities to represent and reason about social commitments and their actions. We present a reduction technique that formally transforms the problem of model checking ACTL^{*c} to the problem of model checking GCTL* (an extension of CTL* with action formulae). We prove that the reduction technique is sound and we fully implement it on top of the CWB-NC model checker to automatically verify the NetBill protocol, a motivated and specified example in the proposed specification language. We also apply the proposed technique to check the compliance of another protocol: the Contract Net protocol with given properties and report and discuss the obtained results. We finally develop a new symbolic algorithm to perform model checking dedicated to the proposed logic.

Keywords: Multi-Agent Systems, Commitment-based Protocols, Reduction, Symbolic Model Checking, Verification

1. Introduction

1.1. Agent Communication and Commitment-based Protocols

Communication among agents is a fundamental aspect in multi-agent systems (MASs). The reason is that most applications of MASs ranging from intelligent transport systems [38], agent-based web services and their communities [4], dynamic

Email addresses: m_elme@encs.concordia.ca (Mohamed El-Menshawy), bentahar@ciise.concordia.ca (Jamal Bentahar), w_elkh@encs.concordia.ca (Warda El Kholy), dssouli@ece.concordia.ca (Rachida Dssouli)

parking negotiation [13], robotic agents [37; 50], manufacturing systems [33] to web service applications [29; 35], have one thing in common: the agents operating in these systems have to communicate. These systems consist of multiple agents that must communicate in order to solve problems. If a problem is particularly complex, large, or unpredictable, the only way it can reasonably be addressed is to develop a number of modular components (agents) which are able to solve a particular problem aspect. Therefore, it is clear that the success of those MASs requires commonly understood languages: *lingua franca* for agents to 'talk' to each other to decide what information to exchange or what action to take as well as powerful mechanisms (protocols) to regulate and coordinate communication among participants within dialogues and conversations.

The developers of the agent communication language of the foundation for intelligent physical agents (FIPA-ACL)[1] have addressed the challenge of incorporating agent communication languages (ACLs) and protocols by proposing a set of conversation protocols, called FIPA-ACL protocols[2]. These protocols can be viewed as specific ACLs designed for particular purposes, such as *Request Interaction protocol*, *English Auction Interaction protocol* and *Contract Net protocol*. For instance, English Auction Interaction Protocol is designed to help auction goods get a higher market price that is recently used in mobile commerce [14]. Technically, FIPA-ACL protocols have succeed in specifying the rules governing interactions and coordinating dialogues among agents by: (1) restricting the range of allowed follow-up communicative acts at any stage during a dialogue; and (2) describing the sequence of messages that FIPA compliant agents can exchange for particular applfications. The MAS community witnessed a shift from FIPA-ACL protocols and traditional computer protocols formalized by making use of *Finite State Machines* [31] and *Petri Nets* [16] to a special class of multi-agent interaction protocols, called commitment-based protocols. In fact, FIPA-ACL protocols are too rigid to be used by autonomous agents because they are specified so that agents must execute them without possibility of handling exceptions that appear at run time, which restricts the protocols' flexibility. Moreover, traditional computer protocols only capture legal orderings of exchanged messages without unduly considering the meanings of those messages [6; 54]. Missing such meanings limits the ability to verify the compliance of agent behaviors with a given protocol [23].

On the other hand, commitment-based protocols are particularly defined in terms of creation and manipulation of social commitments [17; 27; 54]. Conventionally,

[1]See FIPA-ACL specifications (1997,1999,2001,2002), `http://www.fipa.org/repository/aclspecs.php3`

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

a social commitment is made by an agent playing the role of debtor and directed towards another agent playing the role of creditor to bring about a certain condition [43]. Recent communication approaches based on social commitments have seen progress in a variety of areas, such as modeling business processes [18], developing artificial institutions [28], modeling service-oriented computing [45], developing web-based MASs [49], and specifying commitment-based protocols [2; 17; 20; 22; 36; 54; 55]. In particular, commitments support flexible executions that enable agents to exercise their autonomy by reasoning about their actions and making choices [54]. This flexibility is also related to the accommodation of exceptions that arise at run time by offering more alternative computations to handle them [36; 55]. Commitments also provide declarative representations of protocols by focusing on what the message means and minimally constrain how the message is exchanged [54] without over-constraints on the communications [55]. Moreover, they provide a principled basis for checking if the agents are acting according to given protocols [49; 52].

1.2. Research Problem

Specifying commitment-based protocols that ensure flexible interactions is only necessary, but not sufficient to automatically verify their conformance with given properties. In open systems such as MASs, the designers and business process modelers of the system as a whole cannot guarantee that an agent complies with its commitments and protocols. A formal verification is beneficial to help them detect and eliminate errors so that such protocols comply with specifications. It also reduces the development cost and increases confidence on the safety, efficiency and robustness at design time. In contrast to MASs, formal verification techniques for agent communication protocols (e.g., commitment-based protocols) are still in their infancy, due to the more complex nature of protocols and autonomy of agents. Few approaches have been proposed to address the above challenge. Some approaches used: (1) local testing technique [49]; (2) static verification technique [11; 12; 48; 54; 55]; and (3) semi-automatic verification technique [52] to detect the compliant and non-compliant agents at the end of the protocol. Although these approaches have made significant progress, they have been criticized by Artikis et al. [1] as they are inefficiently applicable in open systems, which have a large state space. Also, their specification languages cannot be directly model checked because those languages are not based on temporal logics. Other approaches have defined commitment-based protocols using existing computational logics to be more applicable in today's economy such as e-negotiation [5], cross-organizational business models [47] and business processes [21; 30]. Those protocols have been verified using different model checking techniques. However, those techniques consist in reducing commitments

and their actions to simple abstract structures and types using informal translation-based approaches to be able to use existing model checking tools. Such informal translation-based approaches [5; 20; 21] have the problem of preventing verifying the real and concrete semantics of commitments and related concepts as defined in the underlying logics. Those approaches only provide partial solution to the problem of model checking commitments because they simply reduce commitment modalities into domain variables. This stops distinguishing among various modes of truth such as necessarily true and true in the future. Also, informal translation-based approaches use simple variables [20; 47] and abstract processes [5; 17] that do not account for the meanings of commitments. Moreover, there are no tools supporting the informal translation-based approaches to perform the translation process before the actual verification process is undertaken.

In our previous work [3; 22; 23], we proposed two effective methods to verify the compliance of commitment-based protocols using existing model checkers:

1. Direct method: we developed a new symbolic algorithm to perform model checking CTLC (an extension of computation tree logic (CTL) introduced by Clarke et al. [15] with modalities for commitments and their fulfillment and violation) [3; 23]. This algorithm is fully implemented on top of the MCMAS model checker [34] to automatically check the correctness of commitment-based protocols, given desirable properties.
2. Indirect method: we presented a formal reduction technique [22] to transform the problem of model checking CTLC$^+$ (a combination of CTL with commitment modality) modeling commitment-based protocols into the problem of model checking CTLK (the combination of CTL with the logic of knowledge [40]) so that the use of the MCMAS model checker is possible and into the problem of model checking ARCTL (an extension of CTL with action formulae) [39] so that the extended NuSMV model checker is usable.

Our previous proposals [3; 22; 23] have successfully painted the following picture: the existing model checkers are feasible to verify the modalities of commitments and their fulfillment and violation correctly without losing the intrinsic meaning, which is the main problem in the informal translation-based approaches. Based on that, the present paper advocates indirect method (reduction method), thanks to its easy implementation and its straight integration of other commitment actions, such as withdraw, delegate and assign (we will explain them in details in Section 2), which are not considered in [3; 22; 23]. Other technical differences with our previous proposals will be discussed in Section 2.2. In particular, we aim to: (1) formally specify commitment-based protocols using a new branching-time temporal logic dedicated to commitments and all associated actions; (2) automatically verify commitment-based protocols against given properties using a reduction technique

to an existing model checker; and (3) introduce a new symbolic model checking algorithm dedicated to the proposed logic.

1.3. Overview of the Proposed Approach

Figure 1 gives an overview of the proposed approach, which consists of three parts. In the first part, we develop a new branching-time temporal logic by extending CTL* introduced by Emerson and Halpern [25] with modalities to represent and reason about social commitments and all associated actions. The introduction of a new logic is motivated by the fact that the needed modal connectives for social commitments and associated actions cannot be expressed using CTL [24], LTL [24] or even CTL* connectors. Furthermore, the election of CTL* is motivated by our objective to achieve a more expressive, succinct and convenient specifications because CTL* subsumes LTL and CTL, while our previous proposals [3; 22; 23] only considered CTL. Called ACTL^{*c}, this new logic is particularly used to: (1) express well-formed formulae of commitments and their contents; and (2) formally specify agent's commitment actions. In the second part, we use social commitments

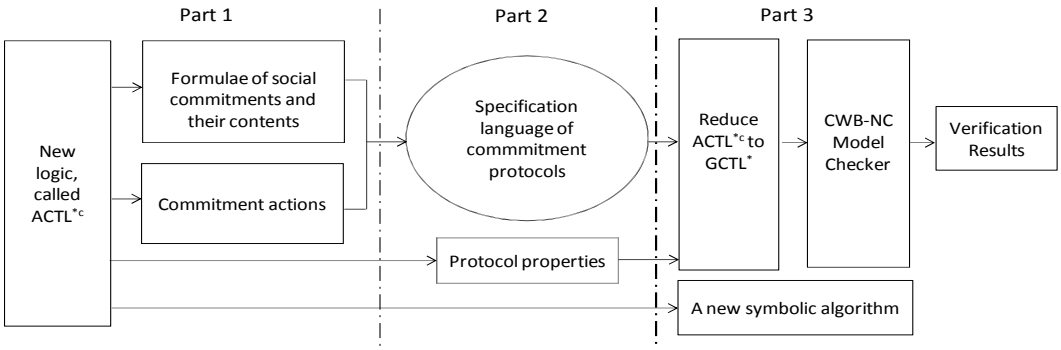

Figure 1: A schematic view of our approach

and associated actions to define a new specification language of commitment-based protocols, which is entirely missing in the literature of agent communication and suitable for model checking. We also use the proposed logic to express some protocol properties. These properties aim to either eliminate unwanted and bad agents' behaviors or enforce desirable agents' behaviors at design time. Furthermore, this paper aims at automatically verifying commitment-based protocols against properties expressed in this part. Such a verification step is implemented in the third part by formally reducing the problem of model checking ACTL^{*c}, representing commitment-based protocols, into the problem of model checking GCTL* [8], an

extension of CTL* with action formulae, so that the use of the CWB-NC automata-based model checker[3] is possible. By formally, we mean that the reduction rules are defined in a systematic and formal way so the correctness (soundness) of the mapping from ACTL^{*c} formulae into GCTL* formulae can be proved. The implemented reduction technique has been used to automatically verify the NetBill protocol [46], a motivated and specified example in the proposed language and the obtained results are reported. In addition to this reduction technique, we also provide a new symbolic model checking algorithm dedicated to ACTL^{*c}. This algorithm provides a methodology to compute the set of states satisfying ACTL^{*c} formulae, which are encoded using Boolean functions that can be easily represented using *Ordered Binary Decision Diagrams* (OBDDs) [9].

The remainder of this paper is organized as follows. Section 2 presents the notion of commitment and the syntax and semantics of the proposed ACTL^{*c}. In Section 3, we use commitments and associated actions to define a new specification language of commitment-based protocols. The reduction of ACTL^{*c} model checking to GCTL* model checking and the theorem that proves the soundness of this reduction technique are discussed in Section 4. The properties of interest to be checked and the verification of the NetBill protocol and the Contract Net protocol using the CWB-NC model checker along with experimental results are also reported in this section. In Section 5, we introduce a new symbolic model checking algorithm for the proposed logic. The paper ends with discussions of relevant literature in Section 6 and conclusion in Section 7.

2. Commitments and ACTL^{*c} Logic

This section presents the first part of our approach, which is about describing the notion of social commitments and defining a new branching-time temporal logic ACTL^{*c}. The commitment formalism defined by Singh and Huhns [45] in their book has the following properties:

Multi-Agency: the agent who promises or commits to bring about some fact is called the debtor and the other agent who wants the fact to be true is called the creditor. Formally, in our approach social commitments are denoted by $C(Ag_1, Ag_2, \varphi)$, where Ag_1 is the debtor, Ag_2 the creditor and φ a well-formed formula (wff) in the proposed logic representing the commitment content. Intuitively, $C(Ag_1, Ag_2, \varphi)$ means Ag_1 socially (i.e., publicly) commits to Ag_2 that φ holds. We can add a new argument to capture deadlines of social commitments as in [7; 48] or define deadlines using temporal quantifiers over intervals as in [36]. In this paper, we use

[3] `http://www.cs.sunysb.edu/cwb/`

the abstract timelines defined in temporal modalities of the proposed logic in order to define deadlines that can be used to manipulate social commitment states (see Section 2.2).

Conditionally: there are some situations where an agent wants to only commit about some facts when a certain condition is satisfied. Formally, we denote conditional commitments by $\tau \supset C(Ag_1, Ag_2, \varphi)$, where "$\supset$" is the logical implication operator, Ag_1, Ag_2 and φ have the above meanings and τ is a wff in our logic representing the commitment condition. We use $CC(Ag_1, Ag_2, \tau, \varphi)$ as an abbreviation of $\tau \supset C(Ag_1, Ag_2, \varphi)$. In this case, we have $C(Ag_1, Ag_2, \varphi) \triangleq CC(Ag_1, Ag_2, \top, \varphi)$ where the propositional constant $\top$ means true.

Manipulability: commitments can be modified in a principled manner using agent's actions, called commitment actions. We classify commitment actions defined in [20; 43] into two and three party actions. The former ones need only two agents to be performed and the latter ones need an intermediate agent to be completed. In the following, we present the intended meaning of these actions and who has the right to perform them.
Two party actions:

- $Withdraw(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$, performed by the debtor Ag_1 to cancel its commitment which is no longer active.
- $Fulfill(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$, when the commitment content is true, the commitment is satisfied.
- $Violate(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$, performed by the debtor Ag_1 to reflect that there is no way to satisfy the commitment.
- $Release(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$, performed by the creditor Ag_2 to free the debtor from carrying out its commitment which is no longer active.

Three party actions:

- $Delegate(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi))$, performed by the debtor Ag_1 to withdraw the current commitment and mark it as delegated to another debtor Ag_3, which creates a new commitment towards Ag_2 with the same content φ in order to satisfy the delegated commitment on behalf of Ag_1.
- $Assign(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi))$, performed by the creditor Ag_2 to release the current commitment and mark it as assigned to another creditor Ag_3, which becomes the creditor of a new commitment of Ag_1 with the same content φ.

Notice that as in business domains, the rights of the debtors and creditors as well as the commitment content are subject to negotiation, which can be performed implicity or explicitly. In this paper, we assume that this negotiation between parties is implicitly performed before the commitment is being established, so it will not be considered in the paper.

2.1. Syntax of ACTL^{*c}

ACTL^{*c} extends CTL* with modalities to represent and reason about social commitments and associated actions. It conceptualizes time as a tree-like structure whose nodes correspond to the states (moments) of the system being considered. The branches or paths of the tree represent all choices in the future that agents have when they participate in conversations, while the past is linear. In what follows, we present the syntax according to which formulae in the proposed ACTL^{*c} can be constructed. Like for CTL*, these formulae are classified into state formulae $\mathcal{S}$ and path formulae $\mathcal{P}$. The state formulae are those that hold on a given state such as commitment formula, while the path formulae express temporal properties of paths and action formulae. The state formulae constitute the formulae of ACTL^{*c}.

Definition 2.1 (Syntax)**.** *The syntax of ACTL^{*c} formulae is given by the following Backus-Naur Form (BNF) grammar:*

$$\begin{aligned}
&\mathcal{S} ::= p \mid \neg\mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid E\mathcal{P} \mid \mathcal{C} \\
&\mathcal{C} ::= C(Agt, Agt, \mathcal{P}) \\
&\mathcal{P} ::= \theta \mid \mathcal{S} \mid \neg\mathcal{P} \mid \mathcal{P} \vee \mathcal{P} \mid \bigcirc \mathcal{P} \mid \mathcal{P}\ U\ \mathcal{P} \mid \alpha(Agt, Agt, \mathcal{C}) \\
&\alpha ::= Wi \mid Fu \mid Vi \mid Re \mid De \mid As
\end{aligned}$$

where:

- $p \in \mathcal{PV}$ *in which* $\mathcal{PV}$ *is a set of atomic propositions;*
- $\theta \in \Phi_\alpha$ *in which* Φ_α *is a set of atomic action propositions;*
- *We use* $\mathcal{A} = \{Ag_1, Ag_2, Ag_3, \ldots\}$ *as a set of agent names. Agt is nonterminal corresponding to the set* $\mathcal{A}$*;*
- *The Boolean operators* $\neg$ *and* $\vee$ *have the usual meaning of negation and disjunction respectively;*
- *The modal connectives* $\bigcirc$ *and* U *are ACTL^{*c} path temporal operators standing for "next time" and "until" respectively;*

– *The universal path quantifier A (read as "for all paths") can be defined using the existential path quantifier E (read as "for some path") and negation:* $A\varphi \triangleq \neg E\neg\varphi$*;*

– *The modal connective* $C(Ag_1, Ag_2, \varphi)$ *stands for social commitment. It is read as "agent* Ag_1 *commits towards agent* Ag_2 *that the path formula* φ *holds" or equivalently as "*φ *is committed to by* Ag_1 *towards* Ag_2*". Committing to path formulae is in fact more expressive than committing to state formulae as state formulae are path formulae and it is not the case that all path formulae are state formulae; and*

– *The modal connective* $\alpha(Ag_1, Ag_2, \mathcal{C})$ *stands for commitment actions, in particular, modal operators* Wi*,* Fu*,* Vi*,* Re*,* De *and* As *stand for Withdraw, Fulfill, Violate, Release, Delegate, and Assign actions respectively. For instance, if* α *is a Delegate action, then* $De(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi))$ *is read as "agent* Ag_1 *delegates its commitment* $C(Ag_1, Ag_2, \varphi)$ *to agent* Ag_3*".*

The remaining operators can be introduced via abbreviations in terms of the above as usual. In particular, $\varphi \wedge \psi \triangleq \neg(\neg\varphi \vee \neg\psi)$ ("conjunction"), $\top \triangleq p \vee \neg p$ ("propositional constant true"), $\bot \triangleq \neg\top$ ("propositional constant false"), $\varphi \supset \psi \triangleq \neg\varphi \vee \psi$ ("material implication" or implication for short), and $\varphi \leftrightarrow \psi \triangleq (\varphi \supset \psi) \wedge (\psi \supset \varphi)$ ("equivalence"). Other modal connectives can be abbreviated in terms of the above as usual, for examples, $\Diamond\varphi \triangleq \top U\varphi$ (eventually) and $\Box\varphi \triangleq \neg\Diamond\neg\varphi$ (globally).

*2.2. Semantics of ACTL*c*

We define the semantics of ACTL*c formula φ with respect to the formal model M associated with commitment-based protocols using a Kripke structure as follows: $M = \langle \mathbb{S}, \mathcal{A}, ACT, R_t, \mathbb{V}_s, \mathbb{V}_\alpha, \mathbb{R}_c, \mathbb{L}, \{\approx_{x,y} \mid (Ag_x, Ag_y) \in \mathcal{A}^2\}, I\rangle$ where:

– $\mathbb{S}$ is a finite set of reachable[4] global states as in the interpreted systems [26] in which each global state is a tuple of local states for all agents in the MAS at a given time,

– $\mathcal{A}$ is a set of agent names where each agent is characterized by a set of possible local states,

– ACT is a set of agent local actions,

– $R_t \subseteq \mathbb{S} \times \mathcal{A} \times ACT \times \mathbb{S}$ is a total labeled transition relation,

[4]This set contains only states that are reachable from I using transition relation R_t.

– $\mathbb{V}_s : \mathcal{PV} \rightarrow 2^{\mathbb{S}}$ is a function assigning to each atomic proposition a set of states satisfying this proposition,

– $\mathbb{V}_\alpha : ACT \rightarrow 2^{\Phi_\alpha}$ is a function assigning to each action a set of atomic action propositions to interpret this action,

– $\mathbb{R}_c : \mathbb{S} \times \mathcal{A} \times \mathcal{A} \rightarrow 2^{\Pi}$, where Π is the set of all paths, is a social accessibility relation for social commitments,

– $\mathbb{L} : \mathbb{S} \rightarrow 2^{\mathcal{A} \times \mathcal{A}}$ is an agency function that associates to each state a set of pairs of two interacting agents in this state, such as the first one is the conveyer and the second one is the addressee,

– $\approx_{x,y} \subseteq \mathbb{S} \times \mathbb{S}$ is an accessibility relation defined by $s_i \approx_{x,y} s_j$ for each pair of agents (Ag_x, Ag_y) iff the local states of Ag_x in the global states s_i and s_j are alternatives *à* la Hintikka's accessibility relation (i.e. indistinguishable) and the same thing for Ag_y such that $(Ag_x, Ag_y) \in \mathbb{L}(s_i)$, and

– $I \subseteq \mathbb{S}$ is a set of initial global states.

We assume that the set ACT of actions includes the special action ϵ for the "*null*" action. Thus, when an agent performs the null action, the local state of this agent remains the same. Furthermore, the underlying time domain in our model M is discrete, i.e., the present moment refers to the current state, the next moment corresponds to the immediate successor state in a given path and a transition corresponds to the advance of a single time-unit. As a modal logic, the time modalities of our logic capture the abstraction view of timelines. This meaning is clarified in the following example.

Example 2.1. *Consider p = deliverGoods, in the context of the NetBill protocol [46]. There, we might want to define a commitment $C(Ag_1, Ag_2,\ E\Diamond^{\leq 7}$ deliverGoods$)$, meaning that a delivery of goods is committed to agent Ag_2 within bounded time, namely 7 days (i.e., time unit is day) by agent Ag_1 where $\Diamond^{\leq 7} p \triangleq p \vee \bigcirc p \vee \bigcirc \bigcirc p \ldots \vee \underbrace{\bigcirc \ldots \bigcirc}_{6\ times} p$.*

Instead of $(s_i, Ag_x, \theta_i, s_{i+1})$, the labeled transitions will be written as $s_i \xrightarrow{Ag_x:\theta_i} s_{i+1}$, which means each transition is labeled with an agent and its action performed during this transition. The paths that path formulae are interpreted over have the form $\pi = \langle s_0 \xrightarrow{Ag_1:\theta_0} s_1 \xrightarrow{Ag_2:\theta_1} s_2 \ldots \rangle$ such that for all $i, x \geq 0, (s_i \xrightarrow{Ag_x:\theta_i} s_{i+1}) \in R_t$. A path in M is then an infinite sequence of reachable global states and labeled transitions with agents and their actions. $\pi(k)$ refers to the k-th state in this

sequence. The set of all paths starting at s_i is denoted by Π^{s_i} while $\langle s_i, \pi \rangle$ refers to the path π starting at s_i. $\pi \uparrow s_i = \langle s_i \xrightarrow{Ag_x:\theta_i} s_{i+1} \xrightarrow{Ag_{x+1}:\theta_{i+1}} s_{i+2} \ldots \rangle$ is the suffix of the path π starting from the state s_i. $\pi \downarrow s_i$ is the prefix of π starting at s_i. When a state s_j is a part of a path π, we write $s_j \in \pi$. Also, when a transition $s_i \xrightarrow{Ag_x:\theta_i} s_{i+1}$ is part of a path π, we write $s_i \xrightarrow{Ag_x:\theta_i} s_{i+1} \in \pi$.

A path $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ is an accessible path for the two interacting agents Ag_1 (the debtor) and Ag_2 (the creditor) iff all global states along this path are reachable and accessible states for the two interacting agents using the agency function, which formally means: $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ iff $s_i = \pi(0)$ and for all $s_j \in \pi$ we have $(Ag_1, Ag_2) \in \mathbb{L}(s_j)$. Intuitively, an accessible path for Ag_1 and Ag_2 from the state s_i is a possible computation of the system for these two agents in the sense of reachability. $\mathbb{R}_c$ has the following properties:

1. $\mathbb{R}_c$ has a form of reflexivity as any accessible path should start from the state itself; i.e., if $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ then $s_i = \pi(0)$ (*axiom T*).
2. If $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ then $\forall s_j \in \pi$ if $\pi' \in \mathbb{R}_c(s_j, Ag_1, Ag_2)$ then $\pi' \downarrow s_j \in \mathbb{R}_c(s_j, Ag_1, Ag_2)$. This property is a form of transitivity (*axiom* 4).
3. If $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ then $\forall s_j \in \pi$ we have $\pi \uparrow s_j \in \mathbb{R}_c(s_j, Ag_1, Ag_2)$.

Thus, the logic of commitment is at least $KT4$, which it is known as $S4$. An accessible state s_j for Ag_1 and Ag_2 from the state s_i (i.e., $s_i \approx_{1,2} s_j$) is an alternative state in the sense that the two local states of each agent in the global states s_i and s_j are indistinguishable and $(Ag_1, Ag_2) \in \mathbb{L}(s_i)$. It is easy to see that $\approx_{x,y}$ is transitive, and if we assume that it is serial (as in any normal modal logic), we obtain that $\approx_{x,y}$ is also reflexive, symmetric, and Euclidean. Thus, the logic of *Withdraw, Fulfill, Violate, Release, Delegate* or *Assign* action is at least $S5$.

Definition 2.2 (Satisfaction)**.** *Satisfaction for an ACTL*^{c} state (resp. path) formula φ in the model M at a global state s_i (resp. along the path π starting at global state s_i), denoted as $M, \langle s_i \rangle \models \varphi$ (resp. $M, \langle s_i, \pi \rangle \models \varphi$), is recursively defined as follows:*

- $M, \langle s_i \rangle \models p$ *iff* $s_i \in \mathbb{V}_s(p)$,
- $M, \langle s_i \rangle \models \neg\varphi$ *iff* $M, \langle s_i \rangle \not\models \varphi$,
- $M, \langle s_i \rangle \models \varphi \vee \psi$ *iff* $M, \langle s_i \rangle \models \varphi$ *or* $M, \langle s_i \rangle \models \psi$,
- $M, \langle s_i \rangle \models E\varphi$ *iff* $\exists \pi \in \Pi^{s_i}$ *s.t.* $M, \langle s_i, \pi \rangle \models \varphi$,
- $M, \langle s_i \rangle \models C(Ag_1, Ag_2, \varphi)$ *iff* $\forall \pi \in \Pi^{s_i}$ *s.t.* $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ *we have* $M, \langle s_i, \pi \rangle \models \varphi$,

- $M, \langle \pi \rangle \models \theta$ *iff* $(\pi(0) \xrightarrow{Ag_x:\beta} \pi(1)) \in R_t$ *and* $\theta \in \mathbb{V}_\alpha(\beta)$ *for an agent* Ag_x,
- $M, \langle s_i, \pi \rangle \models \varphi$ *iff* $M, \langle s_i \rangle \models \varphi$,
- $M, \langle s_i, \pi \rangle \models \neg\varphi$ *iff* $M, \langle s_i, \pi \rangle \not\models \varphi$,
- $M, \langle s_i, \pi \rangle \models \varphi \vee \psi$ *iff* $M, \langle s_i, \pi \rangle \models \varphi$ *or* $M, \langle s_i, \pi \rangle \models \psi$,
- $M, \langle s_i, \pi \rangle \models \bigcirc\varphi$ *iff* $M, \langle s_{i+1}, \pi \uparrow s_{i+1} \rangle \models \varphi$ *on the suffix* $\pi \uparrow s_{i+1}$,
- $M, \langle s_i, \pi \rangle \models \varphi \; U \; \psi$ *iff* $\exists j \geq i$ *s.t.* $M, \langle s_j, \pi \uparrow s_j \rangle \models \psi$ *and* $M, \langle s_k, \pi \uparrow s_k \rangle \models \varphi \; \forall i \leq k < j$,
- $M, \langle s_i, \pi \rangle \models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i \rangle \models \neg C(Ag_1, Ag_2, \varphi)$ *and* $\exists s_j$ *s.t.* $s_i \approx_{1,2} s_j$ *and* $M, \langle s_j \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\pi \notin \mathbb{R}_c(s_i, Ag_1, Ag_2)$,
- $M, \langle s_i, \pi \rangle \models Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\forall s_j$ *s.t.* $s_i \approx_{1,2} s_j$ *we have* $M, \langle s_j \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$,
- $M, \langle s_i, \pi \rangle \models Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\forall s_j$ *s.t.* $s_i \approx_{1,2} s_j$ *we have* $M, \langle s_j \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $M, \langle s_i, \pi \rangle \models \neg\varphi$,
- $M, \langle s_i, \pi \rangle \models Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i \rangle \models \neg C(Ag_1, Ag_2, \varphi)$ *and* $\exists s_j$ *s.t.* $s_i \approx_{2,1} s_j$ *and* $M, \langle s_j \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\pi \notin \mathbb{R}_c(s_i, Ag_1, Ag_2)$,
- $M, \langle s_i, \pi \rangle \models De(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i, \pi \rangle \models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ *and* $M, \langle s_i \rangle \models C(Ag_3, Ag_2, \varphi))$,
- $M, \langle s_i, \pi \rangle \models As(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i, \pi \rangle \models Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$ *and* $M, \langle s_i \rangle \models C(Ag_1, Ag_3, \varphi))$.

Excluding commitment modality and action formulae, the semantics of ACTL^{*c} state formulae is as usual (semantics of CTL*). The state formula $C(Ag_1, Ag_2, \varphi)$ is satisfied in the model M at s_i iff the content φ is true in every accessible path from this state using $\mathbb{R}_c(s_i, Ag_1, Ag_2)$. The intuition behind defining a commitment as a state formula, instead of a path formula, is to reflect the fact that the debtor agent does not know what will be happening in the future along the path. It is worth noticing that the semantics of commitments entirely differs from the ones introduced in our previous frameworks [3; 22; 23] in terms of the definition of accessibility relation, which here generates a set of accessible paths, but in [3; 22; 23] it produces a set of accessible states. This is because in the previous work, we only used an extension of CTL, but in this paper, the new logic is based on the full computation tree logic CTL*.

The formula $Wi(Ag_1, Ag_2, C(Ag_1,\ Ag_2, \varphi))$, which means *Withdraw* action, is satisfied in the model M at s_i through a path π iff the negation of the formula $C(Ag_1,\ Ag_2, \varphi)$ holds at the current state (i.e., s_i), but the commitment holds in a state s_j which can be seen from the current state s_i through the accessibility $\approx_{1,2}$, and π is not accessible (see Figure 2). The intuition we capture by this semantics is that by withdrawing its commitment, the debtor agent selects a non-accessible path where the commitment is no more active, but still there is an accessible state where the commitment is active so it can be manipulated by fulfillment or violation. The semantics of the formula $Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$, which means *Release* action, is defined in the same way. The only difference is, however, in the accessibility relation $\approx_{2,1}$ instead of $\approx_{1,2}$ because in the case of release, the creditor who performs the action.

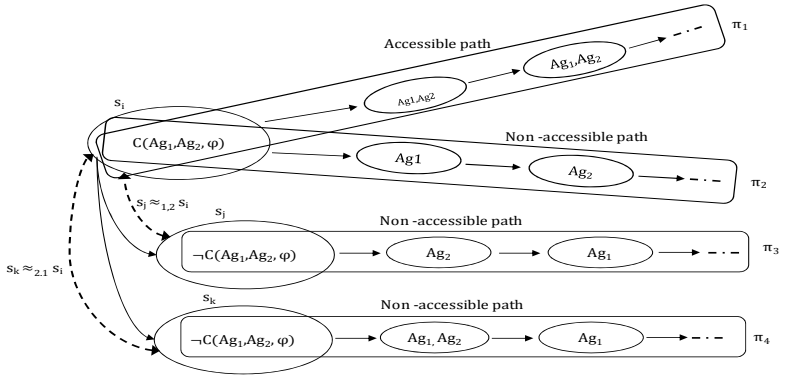

Figure 2: The intuition of the proposed semantics. We have: $M, \langle s_i, \pi_1\rangle \models Fu(Ag_1, Ag_2,\ C(Ag_1, Ag_2, \varphi))$, $M, \langle s_i, \pi_2\rangle \models Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$, $M, \langle s_j, \pi_3\rangle \models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ and $M, \langle s_i, \pi_4\rangle \models Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$.

The formula $Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$, which means *Fulfill* action, is satisfied in the model M at s_i through π iff 1) the commitment $C(Ag_1, Ag_2, \varphi)$ holds both in the current state (i.e., s_i) and every accessible state from it using $\approx_{1,2}$ and 2) the path π starting at the current state is accessible using $\mathbb{R}_c(s_i, Ag_1, Ag_2)$ (see Figure 2). The intuition this semantics captures is that Ag_1 fulfills its commitment if the commitment is still active in all alternative states and Ag_1 selects an accessible path through which the content holds. The commitment is violated in a state along a path π iff the commitment is active in all alternative states and its content is false along the path. Intuitively, when an agent violates its commitment (i.e., the commitment content does not hold), this implies that the agent selects a non-accessible path (see Figure 2), but the contrary is not always true (i.e., being on a non-accessible path does not imply that the content is false along the path).

The semantics of the formulae De and As, which mean *Delegate* and *Assign* actions respectively, is simply defined in terms of the formulae Wi and Re and new commitments. A path π in the model M satisfies action proposition θ iff the label of the first transition on this path satisfies θ.

The notation $\models \varphi$ refers to the validity of the formula φ, which means φ holds for all models and for all states and paths. The following proposition is a direct result from the semantics.

Proposition 2.1.

1. $\models Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$
2. $\models Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$
3. $\models Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$
4. $\models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$
5. $\models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$
6. $\models Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$

3. Commitment-based Protocols

In this section, we proceed to the second part of our approach, which focusses on using the proposed logical model to derive a new specification language of commitment-based protocols. While modeling interactions among agents in terms of commitments provides a good basis for checking compliance, this compliance is only determined by specifying the notion of protocols.

3.1. Protocol Specification

Our specification language of commitment-based protocols is defined using: (1) a set of commitment actions without any constraint; (2) a set of autonomous agents (roles) that communicate by sending messages to each other; and (3) a set of propositions related to the application domain of the protocol. In addition to what and when messages can be exchanged, the proposed specification specifies the meaning of messages in terms of their effects on the commitments. Each message is denoted by $Message(snd, rcv, \psi)$ where snd is the sender, rcv the receiver and ψ (a wff in ACTL^{*c}) the message content representing the exchanged information. This message can be mapped into an action on a commitment in which ψ is mapped into the commitment content. We assume that exchanging messages among agents is reliable, which operationally means messages do not get lost and communication channels are order-preserving. In this manner, a protocol is *public* meaning that it is published and stored in a public repository to be accessible by all participating agents. Notice that our proposed specification can be called a constitutive specification to be compatible with the literature about commitment-based protocol specifications.

The protocol specification begins by a message MSG, which can be followed by other messages (in a recursive way) or ϵ message. This message MSG can be directly mapped into either commitment actions or domain proposition actions. Specifically, *MSG* could either be *Withdraw*, *Fulfill*, *Violate*, *Release*, *Assign*, *Delegate*, *Dom_Pro* or ϵ. These messages are defined in our logic with action formula (α) and atomic action proposition θ respectively. The domain proposition such as *requestQuote*, *refund*, etc. is mainly related to the application domain of the protocol. Each domain application can be represented by a suitable ontology. The formal

Table 1: The formal specification of commitment-based protocols

Protocol	::= *MSG*
MSG	::= $\Big[$*Withdraw*(Ag_1, Ag_2,COM) | *Fulfill*(Ag_1, Ag_2,COM)
	| *Violate*(Ag_1, Ag_2,COM) | *Release*(Ag_2, Ag_1,COM)
	| *Assign*(Ag_2, Ag_3, COM); COM
	| *Delegate*(Ag_1, Ag_3,COM); COM | *Dom_Pro*$\Big]$; $\Big[$*MSG* | ϵ $\Big]$
COM	::= $CC(Ag_1, Ag_2$,Pro,Pro)| $C(Ag_1, Ag_2$,Pro)
Pro	::= A well-formed formula in ACTL^{*c}
Dom_Pro	::= Identify domain propositions

specification language of commitment-based protocols is defined in Table 1 using a Backus-Naur form grammar with meta-symbols: "|" and ";" for respectively the choice, and message sequence that captures the dependence among some messages as for the delegate message that should follow with the new commitment.

3.2. Protocol Compilation

The proposed specification can either be used to reason about the commitment actions at run time [54] or compiled into finite state machines (FSMs) at design time [53]. At run time, the agents can logically compute their execution paths that respect the given protocol specifications using some reasoning rules as axioms. However, these rules are not enough to verify protocols against some properties when the system is large and complex. Moreover, "the flexibility comes at the price of reasoning with declarative representations at run time, which can be expensive and may increase the code footprint of the agents" [44; 54]. For these reasons, we adopt the second approach, which consists in compiling the commitment-based protocol into FSMs where commitments hold on states and actions are labeling transitions (see Figure 3). This is compatible with our ACTL^{*c} logic where commitments are state formulae and actions are path formulae.

As in [44], the proposed protocol specifications are non-terminating analogous to those in real-life applications. In this sense, compiling these protocols requires to

consider Büchi automata, which are automata over infinite words. Büchi automata have the same transition rules like FSMs, but their acceptance condition handles non-terminating paths (or computations) by considering "acceptance states", which are visited infinitely often.

Definition 3.1. *A Büchi automaton is a five tuple* $\mathcal{B} = (Q, \Sigma, Q_0, F, \delta)$ *where* Q *is a set of states;* Σ *is an alphabet;* $Q_0 \in S$ *is a set of initial states;* $F \subseteq Q$ *is a set of accepting states; and* $\delta \subseteq Q \times \Sigma \times Q$ *is the labeled transition relation.*

Definition 3.2. *Let* $inf(\pi)$ *be the set of states that occur infinitely often in the computation (path)* π*. The computation* π *is an accepted run*[5] *by* $\mathcal{B}$ *iff* $inf(\pi) \cap F \neq \emptyset$*.*

3.3. A Motivating Example

Let us consider the NetBill protocol [46] taken from e-business domain to clarify the specification language of commitment-based protocols. The NB protocol is a security and transaction protocol optimized for the selling and delivery of low-priced information goods over the Internet. Figure 3 depicts an extension of this protocol represented as a Büchi automaton. The formal model M (see Section 2.2) of the NetBill protocol is compiled using the Büchi automaton $\mathcal{B}$ defined in Definition 3.1 as follows: Q=$\mathbb{S}$ the set of states (nodes), Σ=$\mathcal{A} \times ACT \cup \{\approx_{Ag_x,Ag_y}\}$ the alphabet used to label transitions, Q_0=I, F=$\{s_6\}$, and δ=$R_t \cup \{\approx_{Ag_x,Ag_y} \mid (Ag_x, Ag_y) \in \mathcal{A}^2\}$ the union of the transition relation and the accessibility relation $\approx_{Ag_x,Ag_y}$.

In particular, our compilation begins by a request quote message on a path starting at s_0 (the transition between s_0 and s_1 is labeled with the *requestQuote* action proposition: $s_0 \xrightarrow{Cus:requestQuote} s_1$). In this message, the customer (*Cus*) requests a price for some desired goods such as software programs. This request is followed by the merchant's (*Mer*) reply by presenting the price quote as an offer. This present quote message generates the *Mer* agent's commitment to deliver the requested goods to the *Cus* agent after receiving the payment ($CC(Mer, Cus, pay, deliver)$) at s_2 (the transition $s_1 \xrightarrow{Mer:presentQuote} s_2$ is labeled with the *presentQuote* action proposition). The *Cus* agent could either reject or accept this offer. Rejecting this offer means the *Cus* agent releases the offer along a path starting at s'_2, which is an equivalent state to s_2 using $\approx_{Cus,Mer}$) (the transition between s'_2 and s_7 is labeled with the *Release* action) and the protocol passes through the failure state s_7 as the computation that cycles through states s_0, s_1, s_2, s'_2, s_7, namely, $(s_0 \xrightarrow{Cus:requestQuote} s_1 \xrightarrow{Mer:presentQuote} s_2 \xrightarrow{Cus:null} s'_2 \xrightarrow{Cus:Release} s_7 \ldots)$ is not accepted by the Büchi automaton, because it does not visit the state s_6 infinitely often. Accepting this offer means the *Cus* agent commits to send the payment to the *Mer*

[5] A run in a Büchi automaton is an infinite sequence of states related by labeled transitions.

agent along another path starting at the state s_2 (the transition between s_2 and s_3 is labeled with the *acceptQuote* action proposition).

Suppose that the *Cus* agent accepts the received offer, then it has three choices: (1) to withdraw its commitment through a path starting at s'_3 (which is equivalent to s_3 using $\approx_{Cus,Mer}$) and passing through s_7, which is not accepted the Büchi automaton; (2) to violate it through a path starting at s_3 and passing through s_7; or (3) to fulfill it by sending the payment to the *Mer* agent along a fourth path starting at the state s_3 but passing through s_4. The computation $(s_3 \xrightarrow{Cus:Fulfill} s_4 \ldots)$ satisfies the formula $Fu(Cus, Mer, C(Cus, Mer, pay))$, i.e., the message *Fulfill* indicates the computation through which the formula is satisfied. In a similar way, the computations $(s_3 \xrightarrow{Cus:Violate} s_7 \ldots)$, and $(s'_3 \xrightarrow{Cus:Withdraw} s_7 \ldots)$ satisfy the formulae $Vi(Cus, Mer, C(Cus, Mer, pay))$, and $Wi(Cus, Mer, C(Cus, Mer, pay))$ respectively.

According to the semantics of *Delegate* action, the *Cus* agent can delegate its commitment to a financial company (say $Bank_1$) to pay the *Mer* agent on its behalf along a path starting at the state s'_3 and passing through s_{11}. As in [52], 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 ($Bank_1$ and $Bank_2$) delegate the commitment back and forth infinitely often and this is presented by transitions that make a loop between states s_{11} and s'_{11}. In a sound protocol, this behavior should be avoided (in Section 4.3, we will show how to verify this issue). The *Mer* agent, before delivering the goods to the *Cus* agent, can withdraw its offer on a path starting at s'_4 (which is equivalent to s_4 using $\approx_{Mer,Cus}$) and passing through s_{10} and then move to the recovery state s_9 (which is not accepted by the Büchi automaton) after refunding the payment to the *Cus* agent, which means performing the *refund* action on a path starting at s_{10}. However, when the *Cus* agent pays for the requested goods and the *Mer* agent delivers them, the *Mer* agent fulfills its commitment along a path starting at the state s_4 and passing through s_5 and then moves to the acceptance state s_6 after sending the receipt to the *Cus* agent along a path starting at s_5. The computation that cycles through states $s_0, s_1, s_2, s_3, s_4, s_5, s_6$, namely, $(s_0 \xrightarrow{Cus:requestQuote} s_1 \xrightarrow{Mer:presentQuote} s_2 \xrightarrow{Cus:acceptQuote} s_3 \xrightarrow{Cus:Fulfill} s_4 \xrightarrow{Mer:Fulfill} s_5 \xrightarrow{Mer:receipt} s_6 \xrightarrow{Mer:null} s_0 \ldots)$ is accepted by the Büchi automaton because it visits the state s_6 infinitely often.

Conversely, the *Cus* agent can pay for the requested goods without being delivered by the *Mer* agent. In this case, the *Mer* agent violates its commitment on another path starting at s_4 and passing through s_8 and then moves to the recovery state s_9 after refunding the payment to the *Cus* agent. As we mentioned, the state s_9 is not accepted by the Büchi automaton. Finally, the *Cus* agent, for some reasons, can assign the commitment of the *Mer* agent to deliver the goods to another

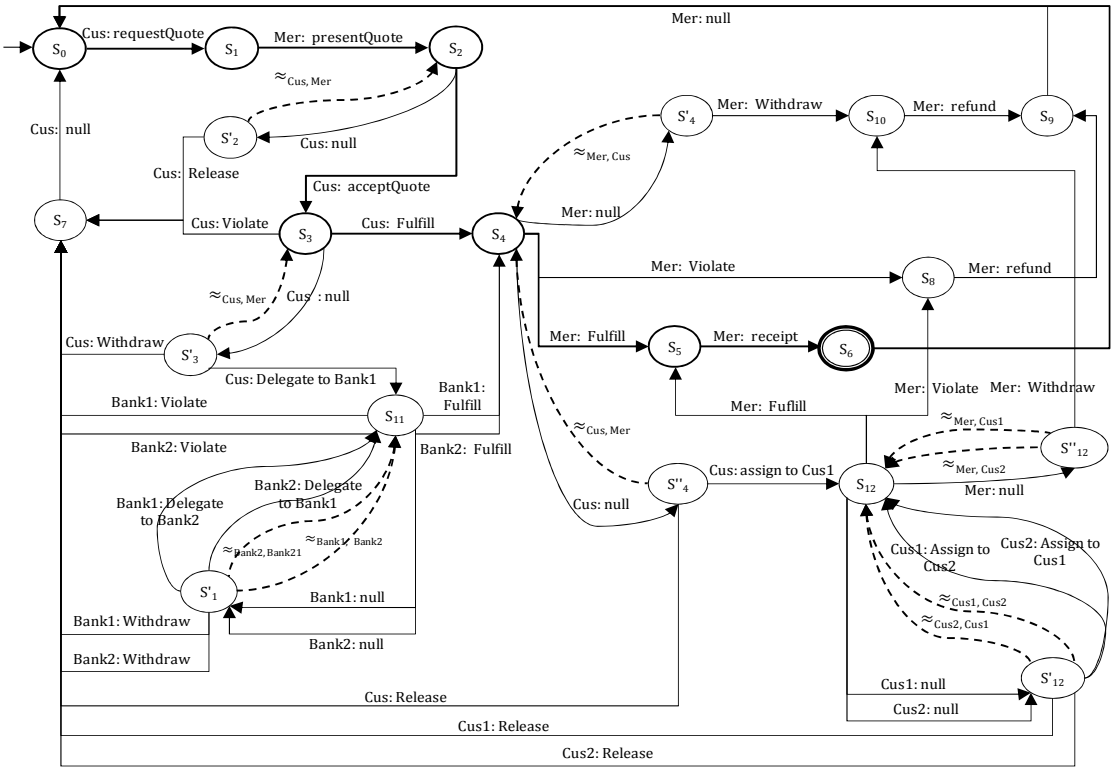

Figure 3: Extension of the NetBill protocol by considering commitments and actions. We notice that the dashed arrows show the accessible states using the accessibility relation $\approx_{Ag_x,Ag_y}$.

customer (say Cus_1) along a fourth path starting at s''_4 (which is equivalent to s_4 using $\approx_{Cus,Mer}$) but passing through s_{12}. Specifically, the *Cus* agent releases the current commitment with the *Mer* agent and a new commitment between *Mer* and Cus_1 is created to deliver the requested goods to the Cus_1 agent. As for delegate scenario, the assign action can be repeated infinitely often among interacting agents and this scenario, presented by transitions that make a loop between states s_{12} and s'_{12}, is unwanted behavior in our protocol.

4. Implementation

The proposed implementation is performed in 4 steps as follows: (1) reduce the problem of model checking ACTL^{*c} into the problem of model checking GCTL* [8] in order to directly use the CWB-NC model checker; (2) encode the NetBill protocol using "Calculus of Communicating Systems (CCS)"–the input language of CWB-NC; (3) express protocol properties; and (4) run the verification of the extended NetBill protocol against the expressed properties and report the experimental results.

4.1. Reducing ACTL^{*c} to GCTL*

GCTL* (read as "Generalized CTL*") extends CTL* [25] by allowing formulae to constrain actions as well as states. The syntax of GCTL* is defined by the following Backus-Naur form grammar [8]:

$$\mathcal{S} ::= p \mid \neg\mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid E\ \mathcal{P}$$
$$\mathcal{P} ::= \theta \mid \mathcal{S} \mid \neg\mathcal{P} \mid \mathcal{P} \vee \mathcal{P} \mid \bigcirc \mathcal{P} \mid \mathcal{P}\ U \mathcal{P}$$

where $p \in \mathcal{PV}$, $\mathcal{PV}$ is a set of atomic propositions and $\theta \in \Phi_\alpha$, Φ_α is a set of atomic action propositions. In fact, this syntax is similar to the syntax of the proposed ACTL^{*c} where the formulae generated by $\mathcal{S}$ and $\mathcal{P}$ are called state formulae and path formulae respectively. State formulae are those that hold on a given state, while path formulae express temporal properties of paths. State formulae constitute the formulae of GCTL*. Other temporal modalities can be defined as abbreviations as usual. To define the semantics of GCTL* formulae, we define the following model.

Definition 4.1 (Model of GCTL*)**.** *A model $M_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ is a tuple where S_G is a nonempty set of states; Ac is a set of actions; $l_S : S_G \rightarrow 2^{\mathcal{PV}}$ is a state labeling function; $l_{Ac} : Ac \rightarrow 2^{\Phi_\alpha}$ is an action labeling function; $\rightarrow \subseteq S_G \times Ac \times S_G$ is a transition relation; and $I_G \subseteq S_G$ is a set of initial states.*

Intuitively, S_G contains the states that the system may enter, and Ac the atomic actions that the system may perform. In this sense, the labeling functions l_S and

l_{Ac} indicate which atomic propositions hold on a given state and action respectively. The GCTL* semantics [8] follows a standard convention in temporal logic, such as CTL*. A state satisfies $A\varphi$ (resp. $E\varphi$) if every path (resp. some paths) emanating from the state satisfies φ. A path satisfies a state formula if the initial state in the path does, while a path satisfies θ if the path contains at least one transition and the label of the first transition on the path satisfies θ. $\bigcirc$ represents the "next-time operator" and has the usual semantics. $\varphi\ U\ \psi$ holds of a path if φ remains true until ψ becomes true.

The reduction process from the problem of model checking ACTL^{*c} to the problem of model checking GCTL* that allows us to a direct use of CWB-NC is defined as follows: given an ACTL^{*c} model $M = \langle \mathbb{S}, \mathcal{A}, ACT, R_t, \mathbb{V}_s, \mathbb{V}_\alpha, \mathbb{R}_c, \mathbb{L}, \{\approx_{x,y} \mid (Ag_x, Ag_y) \in \mathcal{A}^2\}, I\rangle$ and ACTL^{*c} formula φ, we have to define a GCTL* model $M_G = \mathscr{F}(M)$ and a GCTL* formula $\mathscr{F}(\varphi)$ using a transformation function $\mathscr{F}$ such that $M \models \varphi$ iff $\mathscr{F}(M) \models \mathscr{F}(\varphi)$. The model $\mathscr{F}(M)$ is defined as a GCTL* model $M_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ as follows:

- $S_G = \mathbb{S}$, $I_G = I$, and $l_S = \mathbb{V}_s$.
- The set Ac is defined as follows: $Ac = \mathcal{A} \times ACT\ \cup\ \{Ag_{x,y} |\ (Ag_x, Ag_y) \in \mathcal{A}^2\}\ \cup\ \{ACC_{x,y} |\ (Ag_x, Ag_y) \in \mathcal{A}^2\}$ where $\mathcal{A}$ and ACT are already used to label temporal transitions in R_t, $Ag_{x,y}$ and $ACC_{x,y}$ are the actions labeling the transitions defined from the accessibility relations $\mathbb{R}_c$ and $\approx_{x,y}$ respectively. Notice that for each element a of Ac, we have two cases:
  1. $a \in \mathcal{A} \times ACT$, in this case a can be written as $(a_\mathcal{A}, a_{ACT})$ such that $a_\mathcal{A} \in \mathcal{A}$ and $a_{ACT} \in ACT$.
  2. $a \in \{Ag_{x,y} |\ (Ag_x, Ag_y) \in \mathcal{A}^2\}\ \cup\ \{ACC_{x,y} |\ (Ag_x, Ag_y) \in \mathcal{A}^2\}$.
- The function l_{Ac} is then defined as follows:
  1. if $a \in \mathcal{A} \times ACT$, then $l_{Ac}(a) = l_{Ac}((a_\mathcal{A}, a_{ACT})) = \mathbb{V}_\alpha(a_{ACT})$,
  2. if not $l_{Ac}(a) = \emptyset$ (notice that $\emptyset \in 2^{\Phi_\alpha}$).
- The labeled transition relation $\rightarrow$ combines the temporal labeled transition R_t and the accessibility relations $\mathbb{R}_c$ and $\approx_{x,y}$ under the following conditions: for states $s_i, s_j \in \mathbb{S}, (Ag_x, Ag_y) \in \mathcal{A}^2$,
  1. $(s_i, Ag_x, \theta, s_j) \in \rightarrow$ iff $(s_i, Ag_x, \theta, s_j) \in R_t$,
  2. $(s_i, Ag_{x,y}, s_j) \in \rightarrow$ iff $s_i \approx_{x,y} s_j$,
  3. $(s_i, ACC_{x,y}, s_j) \in \rightarrow$ iff $\exists s, \pi \in \mathbb{R}_c(s, Ag_x, Ag_y)$ and $(s_i, -, -, s_j) \in \pi$ where the second and third arguments of $(s_i, -, -, s_j)$ can take whatever value from $\mathcal{A}$ and ACT respectively.

With respect to the proposed semantics, the transformation of an ACTL^{*c} formula into a GCTL* formula is defined as follows:

- $\mathscr{F}(p) = p$, if p is an atomic proposition,
- $\mathscr{F}(\neg\varphi) = \neg\mathscr{F}(\varphi)$, and $\mathscr{F}(\varphi \vee \psi) = \mathscr{F}(\varphi) \vee \mathscr{F}(\psi)$,
- $\mathscr{F}(E \bigcirc \varphi) = E \bigcirc \mathscr{F}(\varphi)$, and $\mathscr{F}(E(\varphi\ U\ \psi)) = E(\mathscr{F}(\varphi)\ U\ \mathscr{F}(\psi))$,
- $\mathscr{F}(E\Box\varphi) = E\Box\mathscr{F}(\varphi)$, and $\mathscr{F}(C(Ag_1, Ag_2, \varphi)) = A(\Box\ ACC_{1,2} \supset \mathscr{F}(\varphi))$,
- $\mathscr{F}(EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(\neg C(Ag_1,Ag_2, \varphi)) \wedge E(Ag_{1,2} \wedge \bigcirc\mathscr{F}(C(Ag_1, Ag_2, \varphi))) \wedge E\Diamond\ \neg ACC_{1,2}$,
- $\mathscr{F}(EFu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(C(Ag_1,Ag_2, \varphi)) \wedge A(Ag_{1,2} \supset \bigcirc\mathscr{F}(C(Ag_1, Ag_2, \varphi))) \wedge E\Diamond\ ACC_{1,2}$,
- $\mathscr{F}(EVi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(C(Ag_1,Ag_2, \varphi)) \wedge A(Ag_{1,2} \supset \bigcirc\mathscr{F}(C(Ag_1, Ag_2, \varphi))) \wedge E(\mathscr{F}(\neg\varphi))$,
- $\mathscr{F}(ERe(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(\neg C(Ag_1,Ag_2, \varphi)) \wedge E(Ag_{2,1} \wedge \bigcirc\mathscr{F}(C(Ag_1, Ag_2, \varphi))) \wedge E\Diamond\ \neg ACC_{1,2}$,
- $\mathscr{F}(EDe(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))) \wedge \mathscr{F}(C(Ag_3, Ag_2, \varphi))$,
- $\mathscr{F}(EAs(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(ERe(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))) \wedge \mathscr{F}(C(Ag_1, Ag_3, \varphi))$.

Notice that we formally reduced commitments and their actions into GCTL* formulae without losing their real and concrete meanings in contrast to when those commitments are represented as simple data structures [17] or domain variables [5; 20; 30; 47]. Moreover, since our reduction is done formally, we can prove its soundness (correctness).

Theorem 4.1 (Correctness)**.** *Let M and φ be respectively an ACTL^{*c} model and formula and let $\mathscr{F}(M)$ and $\mathscr{F}(\varphi)$ be the corresponding model and formula in GCTL*. We have $M \models \varphi$ iff $\mathscr{F}(M) \models \mathscr{F}(\varphi)$.*

Proof. We prove this theorem by induction on the structure of the formula φ:

If φ is a pure CTL* formula, the correctness is straightforward from the fact that GCTL* is also an extension of CTL*.

If φ is not a pure CTL* formula, by induction over the structure of φ, all the cases are straightforward once the following cases are analyzed.

- $\varphi = C(Ag_1, Ag_2, \psi)$. We have $M, \langle s\rangle \models C(Ag_1, Ag_2, \psi)$ iff $M, \langle s, \pi\rangle \models \psi$ for every $\pi \in \Pi^w$ such that $\pi \in \mathbb{R}_c(s, Ag_1, Ag_2)$. Consequently, $M, \langle s\rangle \models C(Ag_1, Ag_2, \psi)$ iff $\mathscr{F}(M), \langle s, \pi\rangle \models \mathscr{F}(\psi)$ for every $\pi \in \mathbb{R}_c(s, Ag_1, Ag_2)$, which means for every π where the transitions are labeled by $ACC_{1,2}$. By semantics of A and θ in GCTL*, we obtain $\mathscr{F}(M), \langle s\rangle \models A(\Box\ ACC_{1,2} \supset \mathscr{F}(\psi))$.

- $\varphi = EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \psi))$. We have $M, \langle s_i, \pi\rangle \models EWi(Ag_1, Ag_2, C($ $Ag_1, Ag_2, \psi))$ iff $M, \langle s_i\rangle \models \neg C(Ag_1, Ag_2, \psi)$ and there exists an accessible state s_j using the accessibility $\approx_{1,2}$ (i.e., $s_i \approx_{1,2} s_j$) where $C(Ag_1, Ag_2, \psi)$ holds and π is not accessible path using $\mathbb{R}_c(s_i, Ag_1, Ag_2)$. By consequently, $M, \langle s_i, \pi\rangle \models EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \psi))$ iff $\mathscr{F}(M)$, $\langle s_i\rangle \models \mathscr{F}$ $(\neg C(Ag_1,$ $Ag_2, \psi))$ and there exists a path such that $(s_i, Ag_{1,2}, s_j) \in \rightarrow$ and $\mathscr{F}(M), \langle s_j\rangle \models$ $\mathscr{F}(C(Ag_1, Ag_2,\ \psi))$ and there exists a path where $\neg ACC_{1,2}$ holds in some future, which makes the path not accessible. By semantics of E, $\bigcirc$, and θ in GCTL*, we obtain $\mathscr{F}(M), \langle s_i, \pi\rangle \models \mathscr{F}(\neg C(Ag_1, Ag_2,\ \psi)) \wedge E(Ag_{1,2} \wedge$ $\bigcirc\mathscr{F}(C(Ag_1, Ag_2, \psi))) \wedge E\Diamond\ \neg ACC_{1,2}$.

- In a similar way, we can complete the correctness prove of our transformation with respect to the other cases: $\varphi = EFu(Ag_1, Ag_2, C(Ag_1, Ag_2, \psi))$, $\varphi = EVi(Ag_1, Ag_2, C(Ag_1, Ag_2, \psi))$, $\varphi = ERe(Ag_2, Ag_1, C(Ag_1, Ag_2, \psi))$, $\varphi = EDe(Ag_1, Ag_3, C(Ag_1, Ag_2, \psi))$, and $\varphi = EAs(Ag_2,\ Ag_3, C(Ag_1, Ag_2, \psi))$, so we are done.

□

4.2. Encoding the NetBill Protocol using CCS

Having defined the reduction process of an ACTL^{*c} model and ACTL^{*c} formula, hereafter we present our encoding of the NetBill protocol, formalized in the model M (see Section 3.3), using the CCS language. The syntax of CCS needed in this paper is defined by the following Backus-Naur form grammar [8]:

$$\mathsf{P} ::= \mathsf{nil} \mid \alpha.\mathsf{P} \mid (\mathsf{P} + \mathsf{P}) \mid (\mathsf{P}|\mathsf{P}) \mid \mathsf{proc}\ \mathcal{C}{=}\mathsf{P}$$

where:

- P refers to the CCS process,
- the process nil means no action whatsoever,
- if P is a process and α is an action prefixing, then α.P is a process,
- if P1 and P2 are processes, then so is P1 + P2 using the choice operator "+",

– if P1 and P2 are processes, then so is P1 | P2 using the parallel composition operator "|", and

– the keyword proc is used to assign the name $\mathcal{C}$ to the process P.

In our encoding using CCS, each agent such as Cus, Mer, Cus_1, Cus_2, $Bank_1$ and $Bank_2$ in the NetBill protocol is represented by a set of processes in a recursive manner. Each process represents an agent's local state. Moreover, a CCS process conceptually uses communication channels to receive messages from other processes using input channels and may send out messages after performing actions to other processes using output channels in a complementary fashion. These channels are reliable guaranteeing timely delivery of messages. Internally, commitment, acceptance, failure and recovery states are defined as variables in the proc statement. The actions of each agent are explicitly represented using atomic action propositions in the proc statement in order to capture the labeled transitions among commitment states. For example, the customer Cus agent can be specified using the CCS language as follows:

proc C_0 ='requestQuote.C_1

proc C_1 =request.presentQuote.C_2

proc C_2 =CC_MerCus_pay_deliver.('acceptQuote.C_3 + ('$Ag_{\mathsf{Cus,Mer}}$.'Release.C_7 + 'null.'Release.C_7))

...

which means the Cus agent initially produces the request quote message and evolves into the state C_1. The Mer agent replies by sending the present quote message, which makes the Cus agent enter into the state C_2. In the state C_2, the Cus agent is willing to produce accept quote message and enter into the state C_3 or the state C_7 after performing: (1) the atomic action $Ag_{Cus,Mer}$ and the action release; or (2) the null and release actions, but before doing this (i.e. choosing between C_3 and C_7), it needs to receive the commitment from the Mer agent as an offer. The encoding of the extended version of the NetBill protocol using the proposed model, as presented in Section 3.3, is available for download[6].

The above encoding of the NetBill protocol is written in the `.ccs` suffixed file and the following protocol properties are first transformed into GCTL* formulae using our reduction tool and then stored in a separate file whose name includes the `.gctl` suffix.

[6] `http://users.encs.concordia.ca/~bentahar/ENB.zip`

4.3. Protocol Properties

Some proposals have been put forward to classify properties that satisfy different requirements of interaction protocols [5; 10; 17; 22; 52]. Yolum [52] verified the correctness of commitment-based protocols at design time with respect to three kinds of generic properties: *effectiveness*, *consistency* and *robustness*. Cheng [10] and Desai et al. [17] classified protocol properties into two classes: *general properties* such as deadlock and termination and *protocol-specific properties* to verify the commitment-based protocols and their composition. Bentahar et al. [5] classified these properties into: *safety*, *liveness* and *deadlock-freedom*. In this paper, we specify a rich class of temporal properties: *fairness*, *safety*, *liveness*, and *reachability* using the proposed logic. These properties include the properties introduced in [10; 17] and satisfy the same functionalities of the properties presented in [52]. However, the focus in this section is on protocol-specific properties as the general properties can be directly checked by the CWB-NC model checker. This means these properties (e.g., safety and liveness) are here functional correctness properties. The differences and similarities of our properties with the properties presented in [10; 17; 52] are explained in Section 6.

– Fairness constraint property. The motivation behind this property is to rule out unwanted behaviors of agents and remove any infinite loop in our protocol. An example of unconditional fairness constraint is given by φ_1, which states that along all paths it is not the case that in the future the $Bank_1$ agent globally delegates the commitment to the $Bank_2$ agent.

$$\varphi_1 = \neg E\Diamond\Box\ \neg De(Bank_1, Bank_2, C(Bank_1, Mer, pay))$$

This constraint allows us avoiding situations such as the bank agents delegate the commitment back and forth infinitely often. 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.

– Safety property. This property means "something bad never happens". In general, it is expressed by $A\Box\ \neg p$ where p characterizes a "bad" situation, which should be avoided. For example, in our protocol a bad situation is: the *Cus* agent fulfills its commitment by sending the payment, but the *Mer* agent never commits to deliver the requested goods:

$$\varphi_2 = A\Box\neg\big(EFu(Cus, Mer, C(Cus, Mer, pay))\ \wedge A\Box\ \neg C(Mer, Cus, deliver)\big)$$

– Reachability property. A particular situation can be reached from the initial state via some computation sequences. For example, along a given path, it is

eventually the case that there is a possibility for the *Mer* agent to fulfill its commitment by delivering the requested goods:

$$\varphi_3 = E\Diamond\ EFu(Mer, Cus, C(Mer, Cus, deliver))$$

– Liveness property. This property means "something good will eventually happen". For example, along all paths it is globally the case that if the *Cus* agent fulfills its commitment by sending the payment, then in all computations in the future the *Mer* agent will either (1) fulfill the commitment by delivering the goods; (2) withdraw this commitment; or (3) violate it:

$$\begin{aligned}\varphi_4 = A\Box\big(&EFu(Cus, Mer, C(Cus, Mer, pay)) \supset \\ &A\Diamond\big[EFu(Mer, Cus, C(Mer, Cus, deliver)) \\ &\vee EWi(Mer, Cus, C(Mer, Cus, deliver)) \\ &\vee EVi(Mer, Cus, C(Mer, Cus, deliver))\big]\big)\end{aligned}$$

The above temporal properties can be generalized to verify the conformance of other multi-agent interaction protocols as they have a corresponding classification in distributed systems to guide protocol designers to check protocol specifications. They are also defined from a general perspective, for example a formula having a future operator and "something good" with respect to the protocol specification can be used to define liveness property.

4.4. Experiments

We implemented the reduction technique on the top of the CWB-NC model checker and provided a thorough assessment of this reduction using two case studies: (1) our extended version of the NetBill protocol; and (2) the Contract Net protocol designed from online business point of view to reach agreements among interacting agents. The contract net protocol is recently used for specifying the negotiation process in multi-agent manufacturing control systems [51] and is used as negotiation protocol in the fuzzy agent-based expert system for steel making process [56]. These case studies were performed on a laptop equipped with the Intel(R) Core(TM) i5-2430M clocked at 2.4GHz processor and 6GB memory running under x86_64 Windows 7.

4.4.1. Verifying the Extended NetBill protocol

Having respectively presented our extended version of the NetBill protocol and its encoding using the CCS process algebra language in Sections 3.3 and 4.2, thereafter we present its automatic verification using the CWB-NC model checker against the safety and reachability properties (cf. Section 4.3) as well as the deadlock property. The deadlock property mainly checks that all the fired transitions are labeled with agents and their actions. Formally, $E\Diamond\neg\{-\}$ [8]. In particular, we conducted 7 experiments, which are reported in Table 2 wherein the number of agents (n); reachable states (#States); transitions (#Trans) and the total time in seconds (i.e., the time for building the model (TBM) plus the time for checking deadlock (TDL), safety (TSF), and reachability (TRE) formulae) are given. These experiments start with a simple business scenario between two agents (*Cus* and *Mer*). This scenario initially begins by the *Cus* agent requests a quote for some goods and the *Mer* agent commits to deliver them after it received the payment where each one of them has possibilities to withdraw and violate its commitment. Thereafter, we start to give the *Cus* agent another possibility to delegate its commitment to the $Bank_1$ agent in experiment 2, which for some reasons delegates its commitment to the $Bank_2$ agent in experiment 3. In experiments 4 and 5, the *Cus* agent can assign its commitment to the Cus_1 agent, which for some reasons assigns its commitment to the Cus_2 agent. By experiment 5, the 6 participating agents in the extended NB protocol are completely modeled.

Table 2: Verification results of the NetBill protocol

n	#States	#Trans	TBM	TDL	TSF	TRE	Total Time
2	267	851	$0.024s$	$0.077s$	$0.064s$	$0.025s$	$0.19s$
3	883	3878	$0.123s$	$0.336s$	$0.254s$	$0.033s$	$0.746s$
4	5293	30206	$1.160s$	$3.154s$	$2.232s$	$0.089s$	$6.635s$
5	18145	121965	$6.031s$	$15.227s$	$10.569s$	$1.033s$	$32.86s$
6	108865	874916	$56.194s$	$138.253s$	$92.316s$	$0.171s$	$286.932s$
7	254017	2302590	$207.231s$	$478.866s$	$312.025s$	$0.190s$	$1684.409s$
8	428545	4288876	$503.171s$	$1119.284s$	$687.520s$	$0.235s$	$5123.356s$

We underline that when the number of reachable states (which reflect the state space of the model) is small, the total time is also small (cf. Exp.1 and Exp.2). However, when the number of reachable states increases (as in Exp.4 and Exp.5), the total time is going to be much higher. In experiment 6, we turned toward showing the effectiveness of our technique in terms of the total time and number of reachable states by making more than one customer request a quote for some goods. For instance, in experiment 6, we have two customers and each one of them

can asynchronously communicate with five agents (2 Bank agents and 3 Merchant agents as we mentioned), so we have 7 participating agents. Also, in experiment 7, we have 3 customers, 2 Bank agents, and 3 Merchant agents, so the total number of participating agents is 8.

4.4.2. Verifying Contract Net Protocol

The Contract Net protocol is already used to show how commitments and their actions can specify protocols in business settings [54; 22]. It starts with a manager requesting proposals for a particular task. Each participant either sends a proposal or a reject message. The manager accepts only one proposal among the received proposals and explicitly rejects the rest. The participant with the accepted proposal informs the manager with the proposal result or the failure of the proposal. As in our modeling of the extended NetBill protocol, we associate the formal model M with the Contract Net protocol and then use the Büchi automaton defined in Definition 3.1 to compile this protocol. The acceptance state is when the participant agent fulfills its commitment by sending result to the manager. The complete encoding of Contract Net protocol using the CCS process algebra language is also available for download[7]

Table 3 reports the verification results of Contract Net protocol against the three properties (safety, reachability and deadlock) using our transformation technique and the CWB-NC model checker on four experiments where the number of agents (n), the number of reachable states (#States), the number of transitions (#Trans) and the total time in seconds (as in Table 2) are given. In the first experiment, we have three agents: the manager agent and two participating agents wherein the first participant can delegate its commitment to the second participant. In experiments 2, 3 and 4, we have one manager agent and respectively 5, 7 and 9 participant agents. It is obvious that when the state space of the model increases, the total time is also going to be much higher.

Table 3: Verification results of Contract Net protocol

n	#States	#Trans	TBM	TDL	TSF	TRE	Total Time
3	721	3040	$0.093s$	$0.030s$	$0.198s$	$0.015s$	$0.336s$
5	1711	8308	$0.325s$	$0.027s$	$0.569s$	$0.018s$	$0.939s$
7	2983	15812	$0.784s$	$0.029s$	$1.268s$	$0.021s$	$2.102s$
9	4537	25552	$1.627s$	$0.035s$	$2.411s$	$0.030s$	$4.103s$

Notice that the three tested formulae are valid (cf. Tables 2 and 3) and the

[7] `http://users.encs.concordia.ca/~bentahar/CNP.zip`

reachability and deadlock formulae show that the CWB-NC automata-based model checking is very powerful in such cases. However, since the safety formula contains a universal path quantifier, it must be performed on the whole model, thereby invalidating the on-the-fly technique implemented in the CWB-NC model checker, which builds the intersection of the two *Alternating Büchi Automata Tableau Automata* (ABTA) of the model and the formula to be checked on demand without exploring the whole model [8]. From our perspective, this is the main cause of a longer time of verification for the given properties having this form (cf. Tables 2 and 3). We can conclude with the proposed technique is efficient and effective in terms of the number of agents and reachable states. These results prove the effectiveness of our reduction tool that allows checking the satisfiability of commitments and their actions as temporal modal connectives and not as domain variables.

5. Symbolic Model Checking Technique

The aim of this section is to complete the third part of our approach by proposing a new symbolic model checking technique dedicated to our ACTL^{*c} logic. This technique alleviates the "state explosion" problem of automata-based techniques as in practice the space requirements for Boolean functions used in symbolic technique are exponentially smaller than for explicit representation. However, the proposed technique cannot totally eliminate the state explosion problem because the state space still increases when the model is getting larger. We start by introducing the problem of ACTL^{*c} model checking.

5.1. Problem Definition

In a nutshell, given the model M representing a commitment-based protocol and a formula φ describing a property, the problem of model checking can be defined as establishing whether or not the model M satisfies φ (i.e., $\forall s \in I : M, \langle s \rangle \models \varphi$). Clarke et al. in their seminal book shown that the problem of CTL* model checking can be reduced to the problem of CTL and LTL model checking (page 48, [15]). The present paper follows a similar approach by effectively reducing the problem of ACTL^{*c} model checking to the problem of model checking ALTL^c and ACTL^c. ALTL^c and ACTL^c are LTL and CTL [15] augmented with modalities for commitments and associated actions. The motivation of this reduction is to use the standard CTL and LTL procedures introduced in [15; 32].

Figure 4 depicts the expressive powers of the main components of our logic in which ACTL^c formulae (e.g., ψ_1 in the figure) are ACTL^{*c} formulae where every occurrence of a path operator is immediately preceded by a path quantifier. Specifically, ACTL^c is obtained from ACTL^{*c} when the following two conditions are used to specify the syntax of path formulae: (1) if φ_1 and φ_2 are state formulae, then

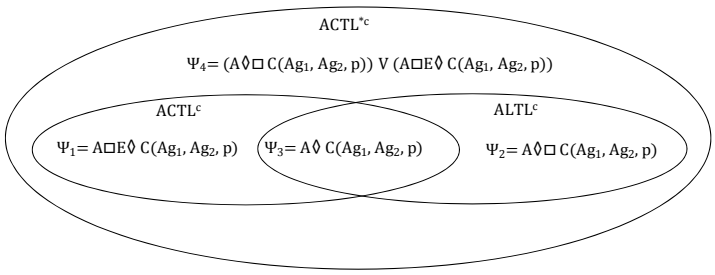

Figure 4: The expressive powers of ACTL^{*c}, ACTL^{c} and ALTL^{c}

$\bigcirc\varphi_1$ and $\varphi_1\ U\ \varphi_2$ are path formulae; and (2) if φ is a path formula, then so is $\neg\varphi$. Whilst, ALTL^{c} formulae (e.g., ψ_2 in the figure) are ACTL^{*c} path formulae in which the permitted state sub-formulae are restricted to atomic propositions. Specifically, a path formula is either: (1) an atomic proposition $p \in \mathcal{PV}$; or (2) it has the form $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $\bigcirc\varphi_1$, and $\varphi_1\ U\ \varphi_2$ where φ_1 and φ_2 are path formulae. The formulae belonging to the intersection (e.g., ψ_3 in the figure) can be expressed in ACTL^{c} and ALTL^{c}. However, the formulae outside the union (e.g., ψ_4 in the figure) are only expressed in ACTL^{*c}, which can be defined as conjunctions or disjunctions of ACTL^{c} and ALTL^{c} formulae so that each sub-formula can be checked using ACTL^{c} or ALTL^{c} model checking.

In general, symbolic model checking techniques are a 3-step process [15; 32]: (1) building the OBDDs of the model M by encoding each element of the model using Boolean functions, which can be represented in OBDDs according to the technique described in [9]; (2) computing the set of states $[\![\varphi]\!]$ satisfying φ in the model and building the OBDD corresponding to this set; and (3) comparing this set $[\![\varphi]\!]$ against the set of initial states I in the model M that is also represented in OBDD: if $I \subseteq [\![\varphi]\!]$, then the model M satisfies the formula; otherwise a counter example can be generated showing why the model does not satisfy the formula. The present paper is concerned with presenting a new symbolic model-checking algorithm to compute the set $[\![\varphi]\!]$ of states satisfying an ACTL^{*c} formula φ (i.e., step 2).

5.2. Symbolic Model Checking Algorithm

The proposed Symbolic Model Checking algorithm $SMC(\varphi, M)$ takes the model M and an ACTL^{*c} formula φ and returns the set of states satisfying the formula φ (i.e., $[\![\varphi]\!]$). We divided our algorithm into main algorithm and sub-algorithms that can be called in the main algorithm. In the main algorithm, we compute the set $[\![\varphi]\!]$ using the following operations on sets: *difference, union, intersection, existential universal quantification.* When sets of states are encoded using Boolean functions, all these operations on sets are translated into operations on Boolean functions

that can be easily represented in OBDDs [9]. For example, the *intersection* of two sets means the *conjunction* of the Boolean functions encoding the two sets. The computation of $[\![\varphi]\!]$ is also based on the use of paths, $\approx$ and $\mathbb{R}_c$, which are encoded as Boolean functions.

The basic idea of our main algorithm is inspired by the algorithm introduced in [15] to verify CTL* formulae using the combination of LTL and CTL model checking algorithms. We extend this algorithm by adding the procedures that deal with the new operators of our logic (see Algorithm 1). As in Clarke et al.'s algorithm [15], we assume that each ACTL^{*c} formula φ can be divided into "maximal state sub-formulae" such that each maximal state sub-formula ψ: (1) includes the existential path quantifier "E"; (2) differs from φ; and (3) is not contained in any other state sub-formula of φ [15]. Each maximal state sub-formula is an ACTL^{*c} formula, which

Algorithm 1 $SMC(\varphi, M)$: the set $[\![\varphi]\!]$ satisfying the ACTL^{*c} formula φ

1: φ is an atomic formula: `return` $\mathbb{V}(\varphi)$
2: φ is $\neg\varphi_1$: `return` $\mathbb{S} \backslash SMC(\varphi_1, M)$
3: φ is $\varphi_1 \vee \varphi_2$: `return` $SMC(\varphi_1, M) \cup SMC(\varphi_2, M)$
4: φ is $C(Ag_1, Ag_2, \varphi_1)$: `return` $SMC_c(Ag_1, Ag_2, \varphi_1, M)$
5: φ is $E\varphi_1$: `return` $SMC_{E\varphi}(\varphi, M)$

can be divided into other maximal state sub-formulae. At level 0, each maximal state sub-formula of φ is identified and at level 1, each of these sub-formulae is divided into other maximal state sub-formulae, and so on until no new maximal state sub-formula can be identified. The algorithm also works in stages such that in stage i all maximal state sub-formulae of φ of level smaller than i are processed (i.e., all states of the model M satisfying these sub-formulae are labeled with them). More precisely, the algorithm starts by checking atomic formula (line 1) and logical operators: negation and disjunction (lines 2 and 3). It then checks the commitment modality (line 4) by calling the procedure $SMC_c(Ag_1, Ag_2, \varphi_1, M)$ (see Algorithm 2). Line 5 deals with an ACTL^{*c} formula that contains the path quantifier E (i.e., $\varphi = E\varphi_1$) (see Algorithm 11).

The procedure $SMC_c(Ag_1, Ag_2, \varphi, M)$ is performed in three steps. In step 1, it computes the set X of states satisfying the formula $E\varphi$ using the standard procedure $SMC_{exi}(E\varphi, M)$ introduced in [32] where φ is the commitment content. The motivation behind computing this set is to ensure that there is a path along which the content holds. In step 2, the procedure builds the set Y of accessible paths from every state in the set X such that the formula φ holds along these paths. The computation of Y is completed by calling the procedure $P\_SAT(\pi, \varphi, M)$ that returns true if the accessible path π satisfies φ and false otherwise (see Algorithm 3). In step 3, the procedure computes the set Z of states satisfying the commitment. Z

Algorithm 2 $SMC_c(Ag_1, Ag_2, \varphi, M)$: the set $[\![C(Ag_1, Ag_2, \varphi)]\!]$

1: $X \leftarrow SMC_{exi}(E\varphi, M)$
2: $Y \leftarrow \{\pi |\ \pi \in \mathbb{R}_c(s, Ag_1, Ag_2)$ and $s \in X$ and $P\_SAT(\pi, \varphi, M)\}$
3: $Z \leftarrow \{s |\ s = \pi(0)$ s.t. $\pi \in Y$ and $\forall \pi' \in \mathbb{R}_c(s, Ag_1, Ag_2) : \pi' \in Y\}$
4: `return` Z

contains all states s at which the accessible paths computed in Y start, such that all accessible paths from s are in Y. This last condition allows excluding from Z the states from which an accessible path emerges, which does not satisfy φ.

Algorithm 3 $P\_SAT(\pi, \varphi, M)$: Boolean

1: φ is an atomic formula: `return` $(\pi(0) \in SMC(\varphi, M))$
2: φ is $\neg\varphi_1$: `return` not$(P_SAT(\pi, \varphi_1, M))$
3: φ is $\varphi_1 \vee \varphi_2$: `return` $P\_SAT(\pi, \varphi_1, M)$ or $P\_SAT(\pi, \varphi_2, M)$
4: φ is $\bigcirc\varphi_1$: `return` $P\_SAT(\pi \uparrow \pi(1), \varphi_1, M)$
5: φ is $\varphi_1\ U\ \varphi_2$: `return` $P\_SAT_U(\pi, \varphi_1, \varphi_2, M)$
6: φ is $Wi(Ag_1, Ag_2, \mathcal{C})$: `return` $P\_SAT_{wi}(\pi, \mathcal{C}, M)$
7: φ is $Fu(Ag_1, Ag_2, \mathcal{C})$: `return` $P\_SAT_{fu}(\pi, \mathcal{C}, M)$
8: φ is $Vi(Ag_1, Ag_2, \mathcal{C})$: `return` $P\_SAT_{vi}(\pi, \mathcal{C}, M)$
9: φ is $Re(Ag_2, Ag_1, \mathcal{C})$: `return` $P\_SAT_{re}(\pi, Ag_2, Ag_1, \mathcal{C}, M)$
10: φ is $De(Ag_1, Ag_3, \mathcal{C})$: `return` $P\_SAT_{de}(\pi, Ag_1, Ag_3, \mathcal{C}, M)$
11: φ is $As(Ag_2, Ag_3, \mathcal{C})$: `return` $P\_SAT_{as}(\pi, Ag_2, Ag_3, \mathcal{C}, M)$

$P\_SAT(\pi, \varphi, M)$ depends on the structure of φ so that when φ is an atomic formula, it returns true if the first state of the path π is in the set of states satisfying φ (i.e., $\pi(0) \in SMC(\varphi, M)$). Otherwise, $P\_SAT(\pi, \varphi, M)$ operates recursively on the structure of φ and calls one of the procedures described in Algorithms 4 to 10. Each procedure takes a path π and the corresponding parameters and returns either

Algorithm 4 $P\_SAT_U(\pi, \varphi, \psi, M)$: Boolean

1: if $(SMC(\psi, M) \cap \{t |\ t \in \pi\} = \emptyset)$ `return` false
2: else $i \leftarrow 0$
3: While $(P\_SAT(\pi \uparrow \pi(i), \varphi, M)$ and $P\_SAT(\pi \uparrow \pi(i), \neg\psi, M))$ do
4: $i \leftarrow i + 1$
5: end while
6: `return` $P\_SAT(\pi \uparrow \pi(i), \psi, M)$
7: end if

true (i.e., the formula holds along the path) or false. For instance, the procedure

$P\_SAT_U(\pi, \varphi, \psi, M)$ (see Algorithm 4) starts by checking whether or not the intersection of the set of states in this path π and the set $[\![\psi]\!]$ is empty. If this is the case, the procedure returns false as ψ will never hold. Otherwise, the procedure iterates over π's transition steps until either φ is not satisfied or ψ becomes true. The procedure returns true if ψ becomes true, and false otherwise.

The procedure $P\_SAT_{wi}(\pi, C(Ag_1, Ag_2, \varphi), M)$ only returns true when there exists an accessible state s from the first state of the path π which is in the set $[\![C(Ag_1, Ag_2, \varphi)]\!]$ and at the same time $\pi(0)$ does not belong to this set and π is accessible for Ag_1 and Ag_2 (see Algorithm 5). In a similar way, the procedure $P\_SAT_{rel}(\pi, C(Ag_1, Ag_2, \varphi), M)$ is reported in Algorithm 6.

Algorithm 5 $P\_SAT_{wi}(\pi, C(Ag_1, Ag_2, \varphi), M)$: Boolean

1: if ($\exists s$ s.t. $\pi(0) \approx_{1,2} s$, $s \in SMC_c(Ag_1, Ag_2, \varphi, M)$) then
2: if $\pi(0) \in \mathbb{S} - SMC_c(Ag_1, Ag_2, \varphi, M)$ and $\pi \in \mathbb{R}_c(\pi(0), Ag_1, Ag_2)$ then
3: `return` true
4: else `return` false
5: end if end if

Algorithm 6 $P\_SAT_{re}(\pi, Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi), M)$: Boolean

1: if ($\exists s$ s.t. $\pi(0) \approx_{2,1} s$, $s \in SMC_c(Ag_1, Ag_2, \varphi, M)$) then
2: if $\pi(0) \in \mathbb{S} - SMC_c(Ag_1, Ag_2, \varphi, M)$ and $\pi \in \mathbb{R}_c(\pi(0), Ag_1, Ag_2)$ then
3: `return` true
4: else `return` false
5: end if end if

The procedure $P\_SAT_{fu}(\pi, C(Ag_1, Ag_2, \varphi), M)$ checks if every accessible state from the first state of the path π and the first state itself are in the set $[\![C(Ag_1, Ag_2, \varphi)]\!]$ and if π is accessible using $\mathbb{R}_c$. In this case, it returns true otherwise, it returns false (see Algorithm 7).

Algorithm 7 $P\_SAT_{fu}(\pi, C(Ag_1, Ag_2, \varphi), M)$: Boolean

1: if ($\forall s$ s.t. $\pi(0) \approx_{1,2} s$, $s \in SMC_c(Ag_1, Ag_2, \varphi, M)$) then
2: if $\pi(0) \in SMC_c(Ag_1, Ag_2, \varphi, M)$ and $\pi \in \mathbb{R}_c(\pi(0), Ag_1, Ag_2)$ then
3: `return` true
4: else `return` false
5: end if end if

The procedure $P\_SAT_{vi}(\pi, C(Ag_1, Ag_2, \varphi), M)$ is similar to the procedure P_SAT_{fu} $(\pi, C(Ag_1,\ Ag_2, \varphi), M)$ except that it does not check the accessibility of π but if π

Algorithm 8 $P\_SAT_{vi}(\pi, C(Ag_1, Ag_2, \varphi), M)$: Boolean

1: if $(\forall s$ s.t. $\pi(0) \approx_{1,2} s$, $s \in SMC_c(Ag_1, Ag_2, \varphi, M))$ then
2: if $\pi(0) \in SMC_c(Ag_1, Ag_2, \varphi, M)$ then
3: `return` $P\_SAT(\pi, \neg\varphi, M)$
4: else `return` false
5: end if end if

satisfies $\neg\varphi$, in which case it returns true, otherwise false (see Algorithm and 8).

According to the proposed semantics, the procedure $P\_SAT_{de}(\pi, Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi), M)$ (reps. $P\_SAT_{as}(\pi, Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi), M)$) starts by checking if the agent Ag_1 (resp. Ag_2) cancels (resp. releases) his commitment along the path π and its first state is in the set $[\![C(Ag_3, Ag_2, \varphi)]\!]$ (reps. $[\![C(Ag_1, Ag_3, \varphi)]\!]$). In this case, the procedure returns true or false otherwise.

Algorithm 9 $P\_SAT_{de}(\pi, Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi), M)$: Boolean

1: if $(P\_SAT_{wi}(\pi, C(Ag_1, Ag_2, \varphi), M))$ and $(\pi(0) \in SMC_c(Ag_3, Ag_2, \varphi, M))$ then
2: `return` true
3: else `return` false
4: end if

Algorithm 10 $P\_SAT_{as}(\pi, Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi), M)$: Boolean

1: if $(P\_SAT_{re}(\pi, Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi), M))$ and
$(\pi(0) \in SMC_c(Ag_1, Ag_3, \varphi, M))$ then
2: `return` true
3: else `return` false
4: end if

The procedure $SMC_{E\varphi}(\varphi, M)$ is the core of our main algorithm as it is responsible for checking the two main cases of $E\varphi$ (see Algorithm 11). In fact, as for CTL* [15], any ACTL^{*c} formula is either ACTL^c formula, ALTL^c formula, a combination of both using $\wedge$ or $\vee$, or a complex nested formula. The case of combined formulae using $\wedge$ or $\vee$ is handled by Algorithm 1 (lines 2 and 3). If the formula is a complex nested formula, then an ALTL^c formula is obtained by replacing each maximal state sub-formula by a new (or fresh) atomic proposition. Algorithm 11 first checks if φ is an ACTL^c formula in that case it calls the procedure $SMC_{actlc}(\varphi, M)$. This procedure is obtained by extending the standard procedure introduced in [32] for CTL formula with the following algorithms: SMC_{Ewi} for $Wi(Ag_1, Ag_2, \mathcal{C})$ (see Algorithm 13), SMC_{Efu} for $Fu(Ag_1, Ag_2, \mathcal{C})$ (see Algorithm 15), SMC_{Evi} for $Vi(Ag_1, Ag_2, \mathcal{C})$

(see Algorithm 16), SMC_{Ere} for $Re(Ag_2, Ag_1, \mathcal{C})$ (see Algorithm 14), SMC_{Ede} for $De(Ag_1, Ag_3, \mathcal{C})$ (see Algorithm 17), and SMC_{Eas} for $As(Ag_2, Ag_3, \mathcal{C})$ (see Algorithm 18). If the formula is not an ACTL^c formula, then each maximal state sub-formula

Algorithm 11 $SMC_{E\varphi}(\varphi, M)$: the set $[\![E\varphi]\!]$

1: φ is an ACTL^c formula: `return` $SMC_{actlc}(\varphi, M)$,
2: $\varphi' \leftarrow \varphi[a_1/E\psi_1, \ldots, a_{\mathcal{K}}/E\psi_{\mathcal{K}}]$,
3: for $i = 1, \ldots, \mathcal{K}$ do
4: $\mathcal{PV} \leftarrow \mathcal{PV} \cup \{a_i\}$ //introducing a new proposition a_i
5: $\mathbb{V}_s(a_i) \leftarrow SMC_{E\varphi}(E\psi_i, M)$ //adding a_i to each state s that satisfies $E\psi_i$
6: end for
7: `return` $SMC_{altlc}(\varphi', M)$

(having the form $E\psi_i$) is replaced by a new atomic proposition (a_i) (line 2), which is added to the set $\mathcal{PV}$ (line 4). Recursively, the set of states satisfying each sub-formula $E\psi_i$ is computed by calling $SMC_{E\phi}(E\psi_i, M)$ (line 5). The procedure is called recursively for the sub-formulae of each level until all the sub-formulae are processed. Finally, the ALTL^c obtained formula φ' is processed and the set of states

Algorithm 12 $SMC_{altlc}(\varphi, M)$: the set $[\![\varphi]\!]$

1: if φ is an LTL formula then `return` $SMC_{ltl}(\varphi, M)$
2: else
3: $\varphi' \leftarrow \varphi[a_1/\mathcal{C}_1, \ldots, a_J/\mathcal{C}_J]$
4: for $i = 1, \ldots, J$ do
5: $\mathcal{PV} \leftarrow \mathcal{PV} \cup \{a_i\}$ //introducing a new proposition a_i
6: $\mathbb{V}_s(a_i) \leftarrow SMC_c(\mathcal{C}_i, M)$ //adding a_i to each state s that satisfies $\mathcal{C}_i$
7: end for
8: `return` $SMC_{altlc}(\varphi', M)$ end if

satisfying it is returned by calling $SMC_{altlc}(\varphi', M)$ (see Algorithm 12). The obtained ALTL^c formula is either an LTL formula, which is handled using the standard LTL model checking [15] (line 1 of Algorithm 12), or a formula containing commitments. In this case, each state commitment formula is replaced by a new atomic proposition (line 3) and the set of states satisfying this formula is computed using $SMC_c(\mathcal{C}_i, M)$ (line 6). The final formula after performing all the replacements in a recursive way is an LTL formula.

The procedure $SMC_{Ewi}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$ first computes the set X (resp. Y) of states that does not satisfy (resp. satisfy) the commitment. It then constructs the set Z of those states (i.e., X) that have accessible states in

Y and from which an accessible path emerges. In a similar way, the procedure $SMC_{Ere}(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi), M)$ is reported in Algorithm 14.

Algorithm 13 $SMC_{Ewi}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))]\!]$

1: $X \leftarrow \mathbb{S} - SMC_c(Ag_1, Ag_2, \varphi, M)$
2: $Y \leftarrow SMC_c(Ag_1, Ag_2, \varphi, M)$
3: $Z \leftarrow \{s \in X |\ \exists s' \in Y$ s.t. $s \approx_{1,2} s'$ and $\exists \pi \notin \mathbb{R}_c(s, Ag_1, Ag_2)\}$
4: `return` Z

Algorithm 14 $SMC_{Ere}(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![ERe(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))]\!]$

1: $X \leftarrow \mathbb{S} - SMC_c(Ag_1, Ag_2, \varphi, M)$
2: $Y \leftarrow SMC_c(Ag_1, Ag_2, \varphi, M)$
3: $Z \leftarrow \{s \in X |\ \exists s' \in Y$ s.t. $s \approx_{2,1} s'$ and $\exists \pi \notin \mathbb{R}_c(s, Ag_1, Ag_2)\}$
4: `return` Z

The procedure $SMC_{Efu}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$ computes the set X_1 of states satisfying the commitment. It then constructs and returns the set X_2 of those states in X_1 that can only see states in the same set using $\approx_{1,2}$ (see Algorithm 15). Notice that we do not need to filter X_2 to only keep states from which an accessible path can emerge because any state satisfies the commitment has such path.

Algorithm 15 $SMC_{Efu}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EFu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))]\!]$

1: $X_1 \leftarrow SMC_c(Ag_1, Ag_2, \varphi, M)$
2: $X_2 \leftarrow \{s \in X_1 |\ \forall s'$ s.t. $s \approx_{1,2} s'$ we have $s' \in X_1\}$
3: `return` X_2

In procedure $SMC_{Evi}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$ (see Algorithm 16), the computation of the sets X_1 and X_2 is defined as in Algorithm 15. The set X_3 contains all states s at which a path π starts such that along this path the negation of the commitment content holds using the procedure P_SAT.

The procedure $SMC_{Ede}(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi), M)$ (resp. $SMC_{Eas}(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi), M)$) computes the set X of states satisfying the delegated (resp. assigned) commitment; then proceeds to build the set Y of states satisfying the withdraw (resp. release) action, see Algorithms 17 and 18 respectively.

Algorithm 16 $SMC_{Evi}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EVi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))]\!]$

1: $X_1 \leftarrow SMC_c(Ag_1, Ag_2, \varphi, M)$
2: $X_2 \leftarrow \{s \in X_1 |\ \forall s'$ s.t. $s \approx_{1,2} s'$ we have $s' \in X_1\}$
3: $X_3 \leftarrow \{s \in X_2 |\ \exists \pi$ s.t. $s{=}\pi(0)$ and $P\_SAT(\pi, \neg\varphi, M)\}$
4: `return` X_3

Algorithm 17 $SMC_{Ede}(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EDe(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi))]\!]$

1: $X \leftarrow SMC_c(Ag_3, Ag_2, \varphi, M)$
2: $Y \leftarrow SMC_{Ewi}(C(Ag_1, Ag_2, \varphi), M)$
3: `return` $X \cap Y$

Theorem 5.1 (Complexity)**.** *The complexity of ACTL^{*c} model checking problem is PSPACE-complete with respect to symbolic representations.*

Sketch. The complexity of ACTL^{*c} depends on the complexity of ACTL^c and ALTL^c. The complexity of ACTL^c depends on the complexity of CTL (PSPACE-complete [42]) and P_SAT algorithm. This algorithm is similar to "PATH" algorithm presented in [41] and its complexity is PSPACE-complete. Here we should notice that the accessibility relations $\mathbb{R}_c$ and $\approx_{x,y}$ only need a polynomial space as only three states should be recorded in memory. From Algorithm 12, the complexity of ALTL^c depends on the complexity of LTL (PSPACE-complete [42]) and Algorithm 2 whose complexity depends also on the complexity of P_SAT algorithm. Consequently, its complexity is PSPACE-complete. □

6. Relevant Work

Yolum and Singh [54] used commitment operations (actions) to show how to flexibly build and execute commitment-based protocols and how to reason about them using the event calculus axioms. In this work, commitment actions are described by a set of axioms on events (or actions happening at time points) and fluents (properties or atomic propositions holding during time intervals). Commitments are modeled as predicates. The authors extended their approach to check if the agent behaviors at run time comply with the given protocol specifications using a static verification technique, called an abductive event calculus planner [55].

Based on Yolum and Singh's representation of commitment actions [55] in terms of the event calculus axioms, Chesani et al. [11] proposed a framework composed of a logical language and a verification procedure. The language defines commitments,

Algorithm 18 $SMC_{Eas}(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EAs(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi))]\!]$

1: $X \leftarrow SMC_c(Ag_1, Ag_3, \varphi, M)$
2: $Y \leftarrow SMC_{Ere}(C(Ag_1, Ag_2, \varphi), M)$
3: `return` $X \cap Y$

deadlines, and "compensations actions" that arise when deadline is expired. The procedure is developed to monitor the status of commitments at run time. Technically, they formalized the event calculus axioms using an extension of abductive logic programming, called $\mathcal{SCIFF}$. The main feature of $\mathcal{SCIFF}$ is that it is event-driven (or reactive) and is based on *reactive event calculus* ($\mathcal{REC}$). In $\mathcal{REC}$ language, commitment and its associated actions are modeled as fluents.

Chopra and Singh [12] specified commitment-based protocols formalized by "non-monotonic commitment machines" using the action logical description language C^+ that formalizes actions with explicit transition systems. In this approach, commitments are modeled as fluents, whereas the intended meaning of two party commitment actions is captured by a set of axioms. As in [55], the authors used a static verification technique in the form of reasoning rules in order to verify the compliance of agent behaviors against a given protocol's state machine.

Desai et al. [19] presented a modular action description geared towards protocols (MAD-P), an extension of the causal logic C^+ to refine and compose protocols from existing ones in order to simplify the development of business processes. This approach specifically enhances Chopra and Singh's approach [12] for representing individual protocols. Commitments are modeled as fluents and the operational semantics of commitment actions is defined as a set of axioms. Composition rules satisfying some requirements are also defined as a set of C^+ axioms without checking the correctness specification of the composed protocol.

Although the above approaches have made significant progress, their specification languages are not suitable for model checking that provides a full automatic verification and is effective in complex systems. Moreover, there is no formal semantics for commitments as they are modeled simply as fluents. And representing commitment actions as axioms or constraints on top of the commitment semantics fails to capture the meaning of interactions that are central to real-life business scenarios and waives the interoperability and verification issues. Our approach defined a new specification language of commitment-based protocols, which can be model checked at design time. This language is derived from a more expressive logic that extends CTL* with commitments and all associated actions modalities.

Venkatraman and Singh [49] developed an approach for locally verifying whether

the behavior of an agent in open systems complies with a commitment-based protocol specified in CTL. Their verification method concentrates on the conditions under which an individual agent may check others' commitments toward itself.

The ideas presented by Venkatraman and Singh were further complemented in two research proposals by [17] and [10]. In particular, the authors in [10; 17] developed the idea of supporting the verification of properties geared toward the composition of commitment-based protocols specified in a particular language called OWL-P. Their properties are specified using LTL to verify the deadlocks and livelocks, where deadlocks can result from the contradiction among composition axioms without considering fairness constraints and reachability properties. They also presented another kind of properties, called "protocol-specific properties", which can be defined using the safety and liveness properties in our approach. Specifically, their verification technique depends on informally translating the protocols into PROMELA (the input language of the SPIN automata-based model checker[8]). However, there is no formal translation tool. They also model commitments as data structures [17] or processes [10]. Gerard and Singh [30] used CTL and the MCMAS model checker[9] to verify protocols refinement that are defined in terms of social commitments without checking the conformance of protocols themselves. They also model commitments as objects and not as modal connectives as we have done here. Representing commitments and their actions as modal operators qualify their truth values. Compared to these approaches [10; 17; 30], our specification language ACTL^{*c} is more expressive and compact than LTL and CTL adopted in [10; 17; 30]. Furthermore, our verification technique is based on formal transformation of commitments and associated actions to GCTL* formulae using reduction method, which is sound.

Yolum [52] presented three generic properties (*effectiveness, consistency* and *robustness*) that are required to develop commitment-based protocols at design time. The effectiveness property is mainly related to check if a given protocol effectively progresses. The consistency property checks that the execution of protocol does not produce conflicting computations. A protocol is robust when the intended tasks can be satisfied by different alternative paths. Our 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 the liveness property and fairness paths accompanied with recovery states that capture the protocol failures, such as if the *Mer* agent

[8] `http://spinroot.com/spin/whatispin.html`.

[9] `http://www-lai.doc.ic.ac.uk/mcmas`.

withdraws or violates its commitment, then it must refund the payment to the *Cus* agent. Finally, the proposed approach enhances Yolum's semi-automatic verification with full-automatic verification using model checking.

The tool CWB-NC is used in [5] to verify the NB protocol without delegation and assignment actions, but exploiting a different encoding of the example and considering only two agents. This encoding models each agent by describing its possible actions and each action is described by a set of states, e.g., *Withdraw* action needs 2 states. There is also a separate process that models the communication channels between two agents. Such an encoding is less efficient than the one presented here, in that the internal states increase the agent model and separated communication channel repeats various parameters for the protocol. While the encoding proposed in [5] allowed for the verification of 2 agents only, the experimental results shown in Table 2 report scenarios with up to 8 agents. The differences with the other proposals can be also discussed at the level of the chapter introduction and motivation. We also presented some comparisons against some proposals through the chapter. While [10; 17; 30] and [5; 20] show that the model checking of commitment-based protocols is feasible, in this work, we focused on formal reduction technique, correctness, efficiency and expressiveness considerations.

7. Conclusion

The novelty of our approach lies in presenting: (1) a new semantics for commitments and all associated actions; (2) a new specification language of commitment-based protocols; (3) a new symbolic model checking algorithm dedicated to ACTL^{*c} logic; and (4) a new technique for verifying commitment-based protocols by reducing the above language without missing or restricting real semantics of commitments and their actions. Using two business protocols, we have experimentally evaluated the effectiveness of our reduction tool and technique along with our verification approach implemented on top of the CWB-NC model checker. These experiments also paint the following picture: the model checker was able to verify a variety of complex formulae correctly and efficiently. Our approach is clearly not exhaustive but helps protocol designers eliminate bad behaviors. This approach establishes the usability of the approach by applying it to a large real-life business processes (approximately $4.2e+06$ states). The overall conclusion coincides with the usual considerations in that automatic verification methods complement current static verification methods very well. When comparing our approach to other approaches in the literature, we find that our approach considerably simplifies the specifications to be checked and maintains the feasibility of the model checking approach.

There are many directions for future work. We plan to continue evaluating this approach by means of other protocols having social semantics so that possible

efficiency advantages may be replicated. We also plan to investigate, from the perspective of logic, protocol specification, and model checking, the relationship between social commitments and other agents' attitudes, particularly knowledge and goals as well as the relationship with conventions.

References

[1] A. Artikis, M. Sergot, and J. Pitt. Specifying Norm-Governed Computational Societies. *ACM Transactions on Computational Logic*, 10(1):1–42, 2009.

[2] M. Baldoni, C. Baroglio, and E. Marengo. Behavior-Oriented Commitment-based Protocols. In H. Coelho, R. Studer, and M. Wooldridge, editors, *ECAI*, volume 215, pages 137–142. IOS Press, 2010.

[3] J. Bentahar, M. El-Menshawy, H. Qu, and R. Dssoulia. Communicative Commitments: Model Checking and Complexity Analysis. *Knowledge-Based Systems*, xxx–xxx:xxx–xxx, 2012. (In Press).

[4] J. Bentahar, Z. Maamar, W. Wan, D. Benslimane, P. Thiran, and S. Subramanian. Agent-based Communities of Web Services: An Argumentation-Driven Approach. *Service Oriented Computing and Applications*, 2(4):219–238, 2008.

[5] J. Bentahar, J.-J. C. Meyer, and W. Wan. Model Checking Agent Communication. In M. Dastani, K. V. Hindriks, and J.-J. C. Meyer, editors, *Specification and Verification of Multi-Agent Systems*, pages 67–102. Springer, 2010.

[6] J. Bentahar, B. Moulin, and B. Chaib-draa. Specifying and Implementing A Persuasion Dialogue Game using Commitments and Arguments. In I. Rahwan, P. Moraitis, and C. Reed, editors, *ArgMAS*, volume 3366 of *LNCS*, pages 130–148. Springer, 2005.

[7] J. Bentahar, B. Moulin, J.-J. C. Meyer, and Y. Lespérance. A New Logical Semantics for Agent Communication. In K. Inoue, K. Satoh, and F. Toni, editors, *CLIMA VII*, volume 4371 of *LNCS*, pages 151–170. Springer, 2007.

[8] G. Bhat, R. Cleaveland, and A. Groce. Efficient Model Checking via Büchi Tableau Automata. In G. Berry, H. Comon, and A. Finkel, editors, *CAV'01*, volume 2102 of *LNCS*, pages 38–52. Springer, 2001.

[9] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[10] Z. Cheng. *Verifying Commitment based Business Protocols and their Compositions: Model Checking using Promela and Spin.* PhD thesis, North Carolina State University, 2006.

[11] F. Chesani, P. Mello, M. Montali, and P. Torroni. Commitment Tracking via the Reactive Event Calculus. In C. Boutilier, editor, *IJCAI*, pages 91–96, 2009.

[12] A. Chopra and M. Singh. Contextualizing Commitment Protocols. In H. Nakashima, M. Wellman, G. Weiss, and P. Stone, editors, *AAMAS*, pages 1345–1352. ACM, 2006.

[13] S.-Y. Chou, S.-W. Lin, and C.-C. Li. Dynamic Parking Negotiation and Guidance using an Agent-based Platform. *Expert Systems with Applications*, 35:805–817, 2008.

[14] Y.-F. Chung, Y.-T. Chen, T.-L. Chen, and T.-S. Chen. An Agent-based English Auction Protocol using Elliptic Curve Cryptosystem for Mobile Commerce. *Expert Systems with Applications*, 38(8):9900–9907, 2011.

[15] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.

[16] R. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Using Colored Petri Nets for Conversation Modeling. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, volume 1916 of *LNCS*, pages 178–192. Springer, 2000.

[17] N. Desai, Z. Cheng, A. Chopra, , and M. Singh. Toward Verification of Commitment Protocols and their Compositions. In E. Durfee, M. Yokoo, M. Huhns, and O. Shehory, editors, *AAMAS'07*, pages 144–146. IFAAMAS, 2007.

[18] N. Desai, A. Chopra, and M. Singh. Amoeba: A Methodology for Modeling and Evolving Cross-Organizational Business Processes. *ACM Transactions on Software Engineering and Methodology*, 19(2):6:1–6:45, 2009.

[19] N. Desai and M. Singh. A Modular Action Description Language for Protocol Composition. In *Proc. of the 22nd AAAI Conference on Artificial Intelligence*, pages 962–967. AAAI Press, 2007.

[20] M. El-Menshawy, J. Bentahar, and R. Dssouli. Verifiable Semantic Model for Agent Interactions using Social Commitments. In M. Dastani, A. E. F. Seghrouchni, J. Leite, and P. Torroni, editors, *LADS'09*, volume 6039 of *LNAI*, pages 128–152. Springer, 2010.

[21] M. El-Menshawy, J. Bentahar, and R. Dssouli. Model Checking Commitment Protocols. In K. G. Mehrotra and et al., editors, *IEA–AIE*, volume 6704 of *LNCS*, pages 37–47. Springer, 2011.

[22] M. El-Menshawy, J. Bentahar, and R. Dssouli. Symbolic Model Checking Commitment Protocols using Reduction. In A. Omicini, S. Sardina, and W. Vasconcelos, editors, *DALT*, volume 6619 of *LNAI*, pages 185–203. Springer, 2011.

[23] M. El-Menshawy, J. Benthar, H. Qu, and R. Dssouli. On the Verification of Social Commitments and Time. In *Proc. of the 10th International Confference on AAMAS*, pages 483–890, 2011.

[24] E. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, chapter 16, pages 995–1072. Elsevier, 1990.

[25] E. Emerson and J. Halpern. Sometimes and NotNever, Revisited: On Branching versus Linear Time Temporal Logic. *Journal of ACM*, 33(1):151–178, 1986.

[26] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. The MIT Press, Cambridge, 1995.

[27] N. Fornara and M. Colombetti. Operational Specification of a Commitment-based Agent Communication Language. In *Proc. of the 1st International Joint Conference on AAMAS*, pages 535–542. ACM, 2002.

[28] N. Fornara, F. Viganò, M. Verdicchio, and M. Colombetti. Artificial Institutions: A Model of Institutional Reality for Open Multi-Agent Systems. *AI and Law*, 16(1):89–105, 2008.

[29] F. García-Sáncheza, R. Valencia-Garcíaa, R. Martínez-Béjarb, and J. Fernández-Breisa. An Ontology, Intelligent Agent-based Framework for the Provision of Semantic Web Services. *Expert Systems with Applications*, 36(2):3167–3187, 2009.

[30] S. Gerard and M. Singh. Formalizing and Verifying Protocol Refinements. *ACM Transactions on Intelligent Systems and Technology*, 2(3):xxx–xxx, 2011. (In Press).

[31] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, New Jersey, 2007.

[32] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Second edition, 2004.

[33] M. Lim and Z. Zhang. A Multi-Agent System using Iterative Bidding Mechanism to Enhance Manufacturing Agility. *Expert Systems with Applications*, 39:8259–8273, 2012.

[34] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *LNCS*, pages 682–688. Springer, 2009.

[35] A. Lomuscio, H. Qu, and M. Solanki. Towards Verifying Contract Regulated Service Composition. *Autonomous Agents and Multi-Agent Systems*, 24(3):345–373, 2012.

[36] A. Mallya, P. Yolum, and M. Singh. Resolving Commitments among Autonomous Agents. In F. Dignum, editor, *ACL'03*, volume 2922 of *LNCS*, pages 166–182. Springer, 2004.

[37] M. Nakano, Y. Hasegawa, K. Funakoshi, J. Takeuchi, T. Torii, K. Nakadai, N. Kanda, K. Komatani, H. Okuno, and H. Tsujino. A Multi-Expert Model for Dialogue and Behavior Control of Conversational Robots and Agents. *Knowledge-Based Systems*, 24(2):248–256, 2011.

[38] S. Parka and V. Sugumaran. Designing Multi-Agent Systems: A Framework and Application. *Expert Systems with Applications*, 28:259–271, 2005.

[39] C. Pecheur and F. Raimondi. Symbolic Model Checking of Logics with Actions. In S. Edelkamp and A. Lomuscio, editors, *Model Checking and Artificial Intelligence*, volume 4428 of *LNCS*, pages 113–128. Springer, 2007.

[40] W. Penczek and A. Lomuscio. Verifying Epistemic Properties of Multi-Agent Systems via Bounded Model Checking. *Fundamenta Informaticae*, 55(2):167–185, 2003.

[41] F. Raimondi. *Model Checking Multi-Agent Systems*. PhD thesis, University College London, 2006.

[42] P. Schnoebelen. The Complexity of Temporal Logic Model Checking. In 4, editor, *Advances in Modal Logic*, pages 1–44, 2002.

[43] M. Singh. An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. *AI and Law*, 7(1):97–113, 1999.

[44] M. Singh. Formalizing Communication Protocols for Multiagent Systems. In M. Veloso, editor, *IJCAI*, pages 1519–1524, 2007.

[45] M. Singh and M. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley, London, 2005.

[46] M. Sirbu. Credits and Debits on the Internet. *IEEE Spectrum*, 34(2):23–29, 1997.

[47] P. Telang and M. Singh. Specifying and Verifying Cross-Organizational Business Models: An Agent-Oriented Approach. *IEEE Transactions on Services Computing*, 4(1):xxx–xxx, 2011. (In Press).

[48] P. Torroni, F. Chesani, P. Mello, and M. Montali. Social Commitments in Time: Satisfied or Compensated. In M. Baldoni, J. Bentahar, M. van Riemsdijk, and J. Lloyd, editors, *DALT*, volume 5948 of *LNCS*, pages 228–243. Springer, 2010.

[49] M. Venkatraman and M. Singh. Verifying Compliance with Commitment Protocols: Enabling Open Web-based Multiagent Systems. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999.

[50] R. Vidoni, F. García-Sánchez, A. Gasparetto, and R. Martínez-Béjar. An Intelligent Framework to Manage Robotic Autonomous Agents. *Expert Systems with Applications*, 38:7430–7439, 2011.

[51] W. Yeung. Behavioral Modeling and Verification of Multi-Agent Systems for Manufacturing Control. *Expert Systems with Applications*, 38:13555–13562, 2011.

[52] P. Yolum. Design Time Analysis of Multi-Agent Protocols. *Data and Knowladge Engineering*, 63(1):137–154, 2007.

[53] P. Yolum and M. Singh. Commitment Machines. In J.-J. C. Meyer and M. Tambe, editors, *ATAL*, volume 2333 of *LNCS*, pages 235–247. Springer, 2002.

[54] P. Yolum and M. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitment. In *Proc. of the Int. Joint Conf. on AAMAS*, pages 527–534. ACM, 2002.

[55] P. Yolum and M. Singh. Reasoning about Commitments in the Event Calculus: An Approach for Sepcifying and Executing Protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):227–253, 2004.

[56] M. Zarandi and P. Ahmadpour. Fuzzy Agent-based Expert Sstem for Steel Making Process. *Expert Systems with Applications*, 36(5):9539–9547, 2009.

Highlights

- Formal semantics for commitments and all associated actions for agent communication.
- Formal specification language of commitment-based protocols.
- Symbolic model checking algorithm dedicated to the proposed logic.
- Reduction technique for the automatic verification of commitment-based protocols.