Auton Agent Multi-Agent Syst (2013) 27:375–418
DOI 10.1007/s10458-012-9208-7

Reducing model checking commitments for agent communication to model checking ARCTL and GCTL*

Mohamed El Menshawy · Jamal Bentahar · Warda El Kholy · Rachida Dssouli

Published online: 30 August 2012

Abstract Social commitments have been extensively and effectively used to represent and model business contracts among autonomous agents having competing objectives in a variety of areas (e.g., modeling business processes and commitment-based protocols). However, the formal verification of social commitments and their fulfillment is still an active research topic. This paper presents CTLC$^+$ that modifies CTLC, a temporal logic of commitments for agent communication that extends computation tree logic (CTL) logic to allow reasoning about communicating commitments and their fulfillment. The verification technique is based on reducing the problem of model checking CTLC$^+$ into the problem of model checking ARCTL (the combination of CTL with action formulae) and the problem of model checking GCTL* (a generalized version of CTL* with action formulae) in order to respectively use the extended NuSMV symbolic model checker and the CWB-NC automata-based model checker as a benchmark. We also prove that the reduction techniques are sound and the complexity of model checking CTLC$^+$ for concurrent programs with respect to the size of the components of these programs and the length of the formula is PSPACE-complete. This matches the complexity of model checking CTL for concurrent programs as shown by Kupferman et al. We finally provide two case studies taken from business domain along with their respective implementations and experimental results to illustrate the effectiveness and efficiency of the proposed technique. The first one is about the NetBill protocol and the second one considers the Contract Net protocol.

Keywords Social commitments · Agent communication · Verification · Reduction

M. El Menshawy (✉) · J. Bentahar · W. El Kholy · R. Dssouli
Faculty of Engineering and Computer Science, Concordia University, Montreal, QC, Canada
e-mail: m_elme@encs.concordia.ca

J. Bentahar
e-mail: bentahar@ciise.concordia.ca

W. El Kholy
e-mail: w_elkh@encs.concordia.ca

R. Dssouli
e-mail: dssouli@ece.concordia.ca

1 Introduction

A society mainly evolves through communication among participating entities. In such a society, people interact and exchange information with each other—despite differences in their languages—to satisfy their individual or social goals that they cannot achieve alone. Such communication requires languages and mechanisms to orchestrate and structure interactions among participants within dialogues.

Correspondingly, an agent-based model for an artificial society should provide acute and adequate means to define a formal semantics for agent communication languages (ACLs). Conventionally, formal semantics lays down the foundation for a neat, concise and unambiguous meaning of agent messages, and provides capabilities to verify if agent behaviors comply with the defined semantics and also facilitates and improves the applicability of the proposed semantics.

In the early days of multi-agent research, a promising way to define ACLs' semantics was widely inspired by Searle's *speech acts theory* [45], which was particulary concerned by identifying actions performed by agents to satisfy their intentions. This is called the *mental approach* that focuses on achieving a rational balance between some concepts of agent communication such as *beliefs, goals, desires* and *intentions*. Under this doctrine, it became apparent that a purely mental semantics of ACLs in terms of pre- and post-conditions of agent's mental states would necessarily impose significant restrictions on the operational behavior of agents. For example, in FIPA-ACL that uses this doctrine, the semantics of *Inform act*, where the sender tells the receiver a proposition p, says that such act can only be uttered if the sender believes the proposition p to be true in the pre-condition part, called sincerity condition part. The problem with this approach is that the addressee agents (or an external observer agent) cannot verify whether the speaker agent violates the pre-conditions defined in such mental semantics or not [49]. This problem is also known as the *semantics verification* problem [60]. Accordingly, such pure mental semantics cannot be used in open systems wherein the interacting agents are *heterogeneous* [49]. Moreover, the semantics of this doctrine makes ACLs not general enough to capture the *interoperability* among heterogeneous systems.

Therefore, there is a shift in multi-agent systems (MASs) community towards *social approaches* to overcome the shortcomings and inconveniences incorporated with mental approaches [47]. Commitments are employed in some of these social approaches, which successfully provide a powerful representation for modeling and representing business contracts among autonomous agents having competing objectives in a variety of areas, such as modeling business processes [18,53,54], developing artificial institutions [27], defining programming languages [59], modeling service-oriented computing [51], developing web-based MASs [57], specifying commitment-based protocols [3,17,40,66], and specifying business protocols [20]. Commitments have also been analyzed in strategic logics such as ATL* where agents can commit to specific strategies [1]. In Longman Dictionary, a commitment is *a promise to do something or to behave in a particular way*. In such a definition, commitment imposes loyalty, dedication or devotion towards a person or group within a community. In broad terms, commitments are social (opposite to psychological commitments [46] or individual commitments [11]), objective and public [16], and help represent the state of affairs in the course of multi-agent interactions. Following social commitment-based approaches, an agent does not need to reason about others' intentions or any agents' mental states.

Singh [48] and Castelfranchi [11] formally denoted social commitments by a 4-argument relation involving a proposition (or an action) p and three agents (i, j, and $ctxt$): $C(i, j, ctxt, p)$, which means i is committed towards j in the social context $ctxt$—which

maybe an agent—to satisfy the proposition p or to do the action p. The agent i that actively makes the social commitment is called the *committer* (or debtor), the agent j to which the commitment is made is called the *committee* (or creditor), and p is called the *content* of the commitment. The context $ctxt$ includes the norms that apply to the group wherein the commitment is established and also resolves disputes between the debtor and creditor [48]. Such context has been removed from the notation of commitments in later proposals coming up from Singh and other researchers. Recently, some authors introduced a different definition where there are a debtor, a creditor, an antecedent condition, and a consequent condition (for example [14,64]). In this definition, a commitment is said to be active when the antecedent condition is true. However, for the sake of simplicity, particularly from the logical perspective, antecedent condition will not be used in this paper, which means we assume that such a condition always holds. More discussion is provided in Sect. 5.

Singh [47] emphasized the need to define the semantics of ACLs in terms of *social notions* by presenting many design advantages of following the social commitment-based approach over the mental one regarding to ACL messages meaning and agent construction in MASs. The author refined his seminal work introduced in [47] with unifying normative concepts and commitments and hence coming up with a rich *descriptive ontology* of commitments [48]. The strong key point in this ontology is the set of actions called commitment actions—that can be used to manipulate commitments—namely, *create, discharge, cancel, release, delegate,* and *assign*. However, for the purpose of model checking, which is the main contribution of this paper, we restrict ourselves only to two actions (fulfillment (or discharge) and violation). We leave the integration of the other actions to our model checking approach to future work.

Singh [49] introduced four crucial criteria that should be satisfied to have a well-defined social semantics for ACL messages:

1. *Formal*: the language must be formal to eliminate the possibility of ambiguity in the meaning of ACL messages and allow agents to reason about them.
2. *Declarative*: the semantics should focus on what the message means instead of how the message is exchanged.
3. *Meaningful*: the semantics should focus on the content of messages, not on their representation as sequences of tokens.
4. *Verifiable*: we can check if the agents are acting according to the given semantics.

Previous approaches have considered defining the semantics of ACL messages in terms of social commitments by means of temporal logics [6–8,21,39,41,49], which we call "logics of commitments". However, the motivation is no longer just formally representing and reasoning about social commitments and commitment actions, but becomes the application of formal and automatic verification techniques, such as model checking in order to verify commitments and commitment-based protocols. It is unrealistic—in open environments (e.g., e-business and e-negotiation)—to assume that all autonomous agents will behave according to the given protocols as they may not behave as they are committed to. Furthermore, a formal verification is necessary to help protocol designers either detect unwanted and bad agents' behaviors to eliminate them or enforce desirable agents' behaviors so that such protocols comply with given specifications at design time.

Several approaches have been widely recognized to address the above challenge. Some approaches used: (1) local testing technique [57]; (2) static verification technique [13,40,55, 66]; and (3) semi-automatic verification technique [63] to identify 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 and Pitt [2] as they are inefficiently applicable in open environments especially with a large state space. 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 [54] and business processes [28]. 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 [5,21] have the problem of preventing verifying the real and concert semantics of commitments and related concepts as defined in the underlying logics. These approaches only provide partial solution to the problem of model checking commitments as they reduce commitment modalities into domain variables. This stops distinguishing among various modes of truth such as necessarily true and true in the future. Furthermore, informal translation-based approaches use simple variables [21,54,12], abstract processes [5,17] that do not account for the meanings of commitments, so we are not checking the defined semantics of commitments. Technically, the notion of commitments has the property of being referentially opaque, which has an opaque context; so substituting the truth values of terms into opaque context is not going to preserve meaning as argued in [61] where it is shown that predicate logics fail in representing the notions of agent's intensions having opaque context. Moreover, there are no tools supporting the informal translation-based approaches to perform the translation process before the actual verification process is undertaken. Thus, we believe that automatic, formal translation-based approach is more suitable as it allows representing commitment modality in other temporal modalities, which can reflect its meaning.

In order for these approaches to address all of the aforementioned challenges, they should satisfy all Singh's criteria within an intergraded framework. Such motivation is recently achieved by means of two methods:

- By a direct method via either developing a proper model checker from scratch or extending an existing model checker with new algorithms for the needed temporal modalities as we did in a previous work for model checking CTLC (an extension of computation tree logic (CTL) introduced by Clarke et al. [15] with modalities for commitments and their fulfillment (i.e., discharge) and violation) [22].
- By a formal reduction method into an existing model checker as we did in another previous work for model checking CTLC [23]. We particulary transformed the problem of model checking CTLC into the problem of model checking CTLK (the combination of CTL with the logic of knowledge [43]) in which the commitment modality is reduced to knowledge modality.

Since our previous proposals [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, the present paper advocates the second method (reduction technique), which is easy to implement and allows to compare different verification techniques with respect to the same logic. We also aim at investigating the effectiveness, and soundness of our approach along with analyzing its computational complexity that is still missing in the existing approaches. In particular, we: (1) introduce CTLC$^+$ by redefining the social accessibility relation introduced in [22,23] to account for the intuition that social commitments are conveyed through communication between agents; (2) refine the semantics of commitments and their fulfillment defined first in [22] (this refinement is designed for the particular purpose of model checking and is functional to the verification at design time); (3) remove violation modality presented in [21,22]; instead we show how to express weak and strong violations as properties in our logic; (4) prove the soundness of the proposed reduction techniques; and (5) analyze the space complexity of CTLC$^+$ model

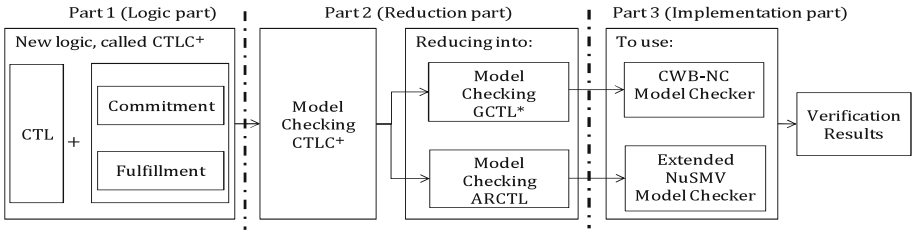

Fig. 1 A schematic view of our approach

checking for concurrent programs, which surprisingly is PSPACE-complete meaning our CTLC$^+$ model checking has the same space complexity as model checking CTL [34] with regard to concurrent programs.

Figure 1 gives an overview of our approach, which consists of three parts. In the first part, we introduce CTLC$^+$, which is a new branching-time temporal logic of commitments including CTL temporal modalities and modalities for commitments and their fulfillment using the formalism of interpreted systems. This logic addresses some limitations in our previous commitment logics [7,8,22] (this aspect will be detailed in Sects. 2.2 and 5). Other commitment actions such as assign, delegate, and withdraw considered in [7,8] from the semantic perspective are not included in this paper for the purpose of being focused on just a subset of these actions, particularly for the purpose of model checking and its computational complexity. Because these actions are needed to take full advantage of the notion of social commitments [18,21,41,48], they will be investigated in our future work.

In the second part, we reduce the problem of model checking CTLC$^+$ into: (1) the problem of model checking ARCTL, the combination of CTL with action formulae [42]; and (2) the problem of model checking GCTL*, an extension of CTL* by allowing formulae to constrain actions as well as propositions [10]. The first reduction allows a direct use of the extended version of the NuSMV model checker introduced in [35]. The second reduction makes using the CWB-NC model checker[1] possible (see Fig. 1). Two reasons have motivated the election of the extended NuSMV and CWB-NC tools: (1) their models are characterized by labeled transitions with actions and during the transformation of our model, these labeled transitions are used to capture the accessibility relations; and (2) they use different model checking techniques namely, ordered binary decision diagrams (OBDDs) implemented in extended NuSMV and alternating Büchi tableau automata (ABTA) implemented in CWB-NC, which gives us the possibility to compare these two techniques with respect to the verification of commitments and their fulfillment. In this part, we also analyze the complexity of model checking CTLC$^+$.

To check the effectiveness of our approach—in the third part (see Fig. 1)—we implement our reduction tools on top of the model checkers (extended NuSMV and CWB-NC) and then report the experimental results of verifying two case studies–widely used to clarify commitment-based protocols: (1) the NetBill protocol; and (2) Contract Net protocol against some desirable properties expressed in our logic.

The remainder of this paper is organized as follows. In Sect. 2, we briefly summarize the formalism of interpreted systems [25] introduced to model MASs and define the syntax and semantics of CTLC$^+$. Reducing the problem of model checking CTLC$^+$ into the problem of model checking ARCTL and GCTL* and the theorems that prove the soundness of our reduction techniques along with complexity analysis of model checking CTLC$^+$ are presented in

[1] http://www.cs.sun\discretionary-ysb.edu/cwb/.

Sect. 3. In Sect. 4, we present two case studies widely studied in agent interactions, the NetBill protocol and Contract Net protocol along with the implementation of our reduction techniques on top of extended NuSMV and CWB-NC. We discuss relevant literature and conclude in Sects. 5 and 6 respectively.

2 Interpreted systems and CTLC$^+$

In this section, we present the first part of our approach, which is mainly related to extend the formalism of interpreted systems introduced by Fagin et al. [25] and modify the temporal logic CTLC introduced in our previous work [22].

2.1 Interpreted systems

The formalism of interpreted systems provides a mainstream framework to model MASs and to explore their fundamental classes such as synchronous and asynchronous. It is also used to express MAS properties in the temporal logic CTL along with the logic of knowledge (or epistemic logic). We advocate this formalism for many reasons summarized as follows:

1. It allows us to abstract from the details of the components and to focus only on modeling key characteristics of the agents and the evolution of their social commitments, which is missing in existing agent communication models.
2. It is a useful tool for ascribing autonomous [36,37] and social behaviors of interacting agents within open MASs [23].
3. It supports the interoperability between global (system) and local (agent) models [23,25].

The formalism of interpreted systems provides a useful framework to locally model autonomous and heterogeneous agents who interoperate within a global system via sending and receiving messages. The concept of local states offers a flexible abstraction for the agents. Technically, local states can be singletons, corresponding to a very high-level description of the agents. However, local states are allowed to have a more complex structure. For instance, local states could be a combination of singletons and a set of variables as we will show later in this paper.

Interpreted systems. Consider a set $\mathcal{A} = \{1, \ldots, n\}$ of n agents and at any given time each agent in the system is in a particular local state. Intuitively, each local state of an agent represents the complete information about the system that the agent has at its disposal at a given moment. We associate a nonempty and countable set L_i of local states for each agent $i \in \mathcal{A}$. To account for the temporal evolution of the system, the formalism of interpreted systems associates with each agent i the set Act_i of local actions. As in interleaved interpreted systems [36], to model synchronous communication among interacting agents, it is assumed that $null \in Act_i$ for each agent i, where $null$ refers to the silence action (i.e., the fact of doing nothing). The action selection mechanism is given by the notion of local protocol $\mathcal{P}_i : L_i \rightarrow 2^{Act_i}$ for each $i \in \mathcal{A}$. That is $\mathcal{P}_i$ is a function giving the set of enabled actions that may be performed by i in a given local state. As in [25], we represent the instantaneous configuration of all agents in the system at a given time via the notion of global state.

Definition 1 ([25]) Let G be the set of all global states and let $g = (l_1, \ldots, l_n)$ be a global state, i.e., $g \in G$ where each element $l_i \in L_i$ represents a local state of agent i, thus the set of all global states $G = L_1 \times \cdots \times L_n$ is the Cartesian product of all local states of n agents.

We use the notation $l_i(g)$ to represent the local state of agent i in the global state g. $I \subseteq G$ is a set of initial global states for the system. As in [25], the global evolution function is defined as follows: $\tau : G \times ACT \rightarrow G$, where $ACT = Act_1 \times \cdots \times Act_n$ and each component $a \in ACT$ is a "joint action", which is a tuple of actions (one for each agent). In addition, the local evolution function τ_i that determines the transitions for an individual agent i between its local states is defined as follows: $\tau_i : L_i \times Act_i \rightarrow L_i$, where if $\tau_i(l_i(g), null) = l_i(g')$ for two given global states g and g', then $l_i(g) = l_i(g')$.

In this paper, we extend the interpreted system formalism to account for communication that occurs during the execution of MAS. This extension allows us to provide an intuitive semantics for social commitments that are established through communication between interacting agents. Thus, we associate with each agent $i \in \mathcal{A}$ a countable set Var_i of local variables that we use to represent communication channels through which messages are sent and received. Each local state $l_i \in L_i$ of agent i is associated with different values obtained from different assignments to variables in Var_i. We denote the value of a variable x in the set Var_i at local state $l_i(g)$ by $l_i^x(g)$. Thus, if $l_i(g) = l_i(g')$, then $l_i^x(g) = l_i^x(g')$ for all $x \in Var_i$. The idea is that, as in distributed systems, for two agents i and j to communicate, they should share a communication channel, which is represented by shared variables between i and j. In this perspective, a communication channel between i and j does exist iff $Var_i \cap Var_j \neq \emptyset$. For a variable $x \in Var_i \cap Var_j$, $l_i^x(g) = l_j^x(g')$ means the values of x in $l_i(g)$ for i and in $l_j(g')$ for j are the same. This intuitively represents the existence of a communication channel between i (in g) and j (in g') through which the variable x has been sent by one of the two agents to the other, and as a consequence of this communication, i and j will have the same value for this variable (see Fig. 2). It is worth noting that shared variables only motivate the existence of communication channels, not the establishment of communication. This aspect is addressed next.

The semantics of our modal language is interpreted using a model generated from the interpreted system formalism. In fact, this model, as in [25], moves away from Kripke models while still benefiting from most of its technical apparatus.

Definition 2 (*Models*) A model $\mathcal{M}_C$=$(S, I, R_t, \{\sim_{i\rightarrow j} \mid (i, j) \in \mathcal{A}^2\}, \mathcal{V})$ that belongs to the set of all models $\mathbb{M}$ is a tuple, where:

- $S \subseteq L_1 \times \cdots \times L_n$ is a set of reachable[2] global states for the system.
- $I \subseteq S$ is a set of initial global states for the system.
- $R_t \subseteq S \times S$ is a transition relation defined by $(s, s') \in R_t$ iff there exists a joint action $(a_1, \ldots, a_n) \in ACT$ such that $\tau(s, a_1, \ldots, a_n) = s'$.
- For each pair $(i, j) \in \mathcal{A}^2$, $\sim_{i\rightarrow j} \subseteq S \times S$ is a social accessibility relation defined by $s \sim_{i\rightarrow j} s'$ iff 1) $l_i(s) = l_i(s')$ and 2) $Var_i \cap Var_j \neq \emptyset$ such that $\forall x \in Var_i \cap Var_j$ we have $l_i^x(s) = l_j^x(s')$ and $\forall y \in Var_j - Var_i$ we have $l_j^y(s) = l_j^y(s')$. We also assume that for any pair $i, j \in \mathcal{A}$, we have that for any $s \in S$, $\sim_{i\rightarrow j}(s) \neq \emptyset$ where $\sim_{i\rightarrow j}(s)$ is the set of accessible states from s, i.e., $\sim_{i\rightarrow j}(s) = \{s' \in S \mid s \sim_{i\rightarrow j} s'\}$.
- $\mathcal{V} : S \rightarrow 2^{\Phi_p}$ is a valuation function where Φ_p is a set of atomic propositions.

The social accessibility relation $\sim_{i\rightarrow j}$ from a global state s to another global state s' ($s \sim_{i\rightarrow j} s'$) captures the intuition that there is a communication channel between i and j ($Var_i \cap Var_j \neq \emptyset$) and s' is a resulting state from this communication initiated by i at s. The channel is thus filled in by i in s, and in s' j receives the information (i.e., the channel's content), which makes the values of all shared variables the same for i and j ($l_i^x(s) = l_j^x(s')$ $\forall x \in$

[2] S contains only states that are reachable from I by means of R_t.

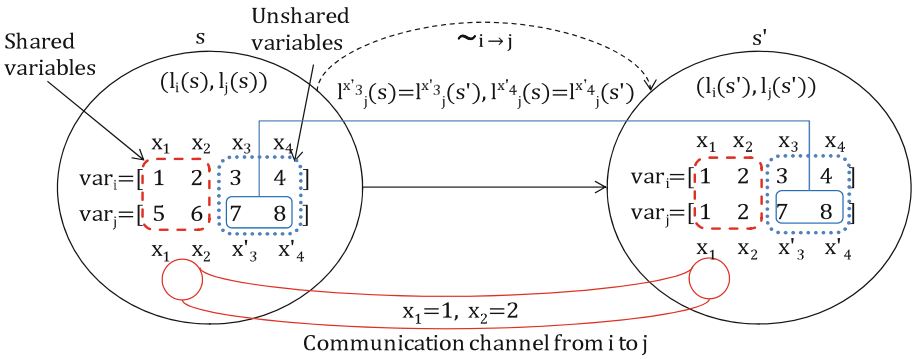

Fig. 2 An example of social accessibility relation $\sim_{i\to j}$. In the example above, the shared and unshared variables for agents are composed as follows. Agent i: $Var_i = \{x_1, x_2, x_3, x_4\}$. Agent j: $Var_j = \{x_1, x_2, x'_3, x'_4\}$. In the figure x_1 and x_2 are shared variables (i.e., they represent the communication channel), and x_3, x_4, x'_3 and x'_4 are unshared variables, which may represent communication channels with other agents. Notice that the values of the variables x_1 and x_2 for j in the state s' are changed to be equal to the values of these variables for agent i, which illustrates the massage passing through the channel. On the other hand, the values of the variables in Var_i are unchanged as $l_i(s) = l_i(s')$

$Var_i \cap Var_j$). As i is the agent who initiates the communication, the source and target (or resulting) states s and s' are indistinguishable for i ($l_i(s) = l_i(s')$). And as j is the receiver, s and s' are indistinguishable with regard to the variables that have not been communicated by i, i.e., unshared variables ($l^y_j(s) = l^y_j(s')\ \forall y \in Var_j - Var_i$). Finally, as our focus in this paper is agent communication, we assume the existence of communication channels between any two agents in any state, which motivates the assumption that $\sim_{i\to j}(s) \neq \emptyset$. This social accessibility relation formalized in terms of relations over the states of a model differs from the ones presented in [22,23] in terms of considering shared and unshared variables. We illustrate this idea in Fig. 2.

In fact, our modeling can be seen as an abstraction of *message-passing systems* described in [25]. Specifically, in message-passing systems, process's local state contains information including internal actions that can change the value of a variable and the messages that it has sent and received. So each agent can directly control and manage communication channels by means of its actions. Furthermore, our extension of the interpreted system formalism by using variables for communication is, to some extent, related to the *modular interpreted systems* approach proposed in [29] where variables are also used for communication purposes. Specifically, modular interpreted systems include a component $\mathcal{I}n$, which represents interaction alphabet and two interaction functions: (1) out_i that illustrates the influence that an action of an agent i may have on the external world; and (2) in_i that illustrates the influences of other agents on the agent i depending on its local state. However, unlike modular interpreted systems, our extension is not meant to focus on the influences of agents through interaction, but on the existence of a communication channel through which two agents can communicate. While communication is the first class citizen in our perspective, the direct interactions are modeled by means of social commitments.

Modeling complex and open systems such as MASs using the formalism of interpreted systems is typically conducted by using logic-based formalisms as formal tools to express desirable properties [25,36,37].

2.2 CTLC$^+$ logic

The syntax of CTLC$^+$ logic, which is a combination of branching time CTL introduced by Clarke et al. [15] with modalities for social commitments and their fulfillment, is defined as follows:

Definition 3 (*Syntax of CTLC$^+$*)

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid E(\varphi\ U\ \varphi) \mid EG\varphi \mid C_{i\rightarrow j}\varphi \mid Fu(C_{i\rightarrow j}\varphi)$$

where:

- $p \in \Phi_p$ is an atomic proposition;
- E is the existential quantifier on paths;
- X, G, and U are CTL path modal connectives standing for "next", "globally", and "until" respectively;
- The Boolean connectives $\neg$ and $\vee$ are defined in the usual way; and
- The modal connectives $C_{i\rightarrow j}$ and Fu stand for "commitment" and "fulfillment of commitment" respectively.

In this logic, $C_{i\rightarrow j}\varphi$ is read as "agent i commits towards agent j that φ", or equivalently as "φ is committed to by i towards j", or simply as "φ is committed to" when i and j are understood from the context. $Fu(C_{i\rightarrow j}\varphi)$ is read as "the commitment $C_{i\rightarrow j}\varphi$ is fulfilled". The constants $\top$ (true) and $\bot$ (false), and other Boolean connectives $\wedge$, $\rightarrow$, and $\leftrightarrow$ are defined as abbreviations in the standard way. Other temporal modalities are given by their usual abbreviations (see for example [15]). In particular, $EF\varphi \equiv E(\top\ U\ \varphi)$, $AX\varphi \equiv \neg EX\neg\varphi$, $AG\varphi \equiv \neg EF\neg\varphi$, and $A(\varphi\ U\ \psi) \equiv \neg E((\neg\psi)\ U\ (\neg\varphi \wedge \neg\psi)) \wedge \neg EG\neg\psi$, where F stands for future. Furthermore, we assume that the underlying time domain in our model $\mathcal{M}_C$ 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. Thus, for example if agent i commits to send goods to agent j within k days (i.e., time unit is day), this would be expressed as $C_{i\rightarrow j}\ EF^{\leq k}\ sendGoods$, where $EF^{\leq k}p \triangleq p \vee EXp \vee EXEXp \cdots \vee \underbrace{EX \cdots EX}_{k\ times} p$.[3] This formulation is motivated by the fact that we are interested in model checking at the design time, so the model should be completely known as the one resulting, for example, from compiling *commitment machines* into FSMs [64]. Thus, it is worth noting that our modeling of commitments does not consider persistency and only focuses on commitments for agent communication.

Computation paths. A computation path $\pi = (s_0, s_1, \ldots)$ in $\mathcal{M}_C$ is an infinite sequence of global states in S such that for all $i \geq 0$, $(s_i, s_{i+1}) \in R_t$. $\pi(k)$ is the kth global state of the path π. The set of all paths is denoted by Π, whilst $\Pi(s_i)$ is the set of all paths starting at the given state $s_i \in S$.

[3] The operator $EF^{\leq k}$ is also used in the real-time CTL logic (RTCTL) introduced in [24].

Definition 4 (*Satisfaction*) Given the model $\mathcal{M}_C$, the satisfaction of a CTLC$^+$ formula φ in a global state s, denoted by $(\mathcal{M}_C, s) \models \varphi$ is recursively defined as follows:

$(\mathcal{M}_C, s) \models p$	iff	$p \in \mathcal{V}(s)$,
$(\mathcal{M}_C, s) \models \neg\varphi$	iff	$(\mathcal{M}_C, s) \not\models \varphi$,
$(\mathcal{M}_C, s) \models \varphi \vee \psi$	iff	$(\mathcal{M}_C, s) \models \varphi$ *or* $(\mathcal{M}_C, s) \models \psi$,
$(\mathcal{M}_C, s) \models EX\varphi$	iff	there exists a path π starting at s such that $(\mathcal{M}_C, \pi(1)) \models \varphi$,
$(\mathcal{M}_C, s) \models E(\varphi U \psi)$	iff	there exists a path π starting at s such that for some $k \geq 0$, $(\mathcal{M}_C, \pi(k)) \models \psi$ and $(\mathcal{M}_C, \pi(j)) \models \varphi$ for all $0 \leq j < k$,
$(\mathcal{M}_C, s) \models EG\varphi$	iff	there exists a path π starting at s such that $(\mathcal{M}_C, \pi(k)) \models \varphi$ for all $k \geq 0$,
$(\mathcal{M}_C, s) \models C_{i\to j}\varphi$	iff	for all global states $s' \in S$ such that $s \sim_{i\to j} s'$, we have $(\mathcal{M}_C, s') \models \varphi$,
$(\mathcal{M}_C, s) \models Fu(C_{i\to j}\varphi)$	iff	there exists $s' \in S$ such that $(\mathcal{M}_C, s') \models C_{i\to j}\varphi$and $s' \sim_{i\to j} s$.

For the propositions, Boolean connectives and temporal modalities, the relation $\models$ is defined in the standard manner (see for example [15]). The state formula $C_{i\to j}\varphi$ is satisfied in the model $\mathcal{M}_C$ at s iff the content φ holds in every accessible state obtained by the accessibility relation $\sim_{i\to j}$. In the specific context of agent communication, which is the focus of this paper, when i commits towards j that φ, $\sim_{i\to j}$ captures the intuition that for a commitment to take place, a communication channel should exist between the communicating agents (shared variables), and the accessible state s' is indistinguishable from the current state s for i since i is the agent who is committing; however, for j who is receiving the commitment, the two states are different as new information are obtained from i through the communication channel and this is why in the accessible state, j has the same values as i has for the shared variables (i.e. the content of the communication channel). Furthermore, the accessible state is not completely different from the current state for j as some information are still the same, and this is why for the unshared variables, the current and accessible states for j are indistinguishable (see Fig. 2).

The state formula $Fu(C_{i\to j}\varphi)$ is satisfied in the model $\mathcal{M}_C$ at s iff there exists a state s' satisfying the commitment (called here the *commitment state*) from which the current state (i.e., s) is "seen" via the accessability relation $\sim_{i\to j}$. The idea behind this semantics is to say that a commitment is fulfilled when we reach an accessible state from the commitment state. The commitment is fulfilled because its content holds in this accessible state. Unlike the semantics proposed in [5,21–23] in which the state s should not only be accessible but also reachable from the commitment state s', in our semantics, the reachability condition is omitted. In fact, being reachable from the commitment state is not a part of the meaning of fulfilling a commitment, but a condition that can be checked as a property as follows:

$$AG\big(\neg E(\neg C_{i\to j}\varphi \; U \; (\neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi)))\big)$$

The property is a condition saying that in all states of every computation, it is not the case that there is a computation where fulfilling a commitment occurs in its future without such a commitment has been established before. What is more interesting is that this property, which guarantees that a commitment cannot be fulfilled without being active (i.e. established or created), is satisfied in every model, so valid (the validity proof is given later in this section). The main advantage of having a property that can be checked (or proved as validity) instead of adding it as a part of the semantics is to simplify such a semantics, which means making its model checking simpler. This is extremely important as far as time and space complexity of model checking is an issue.

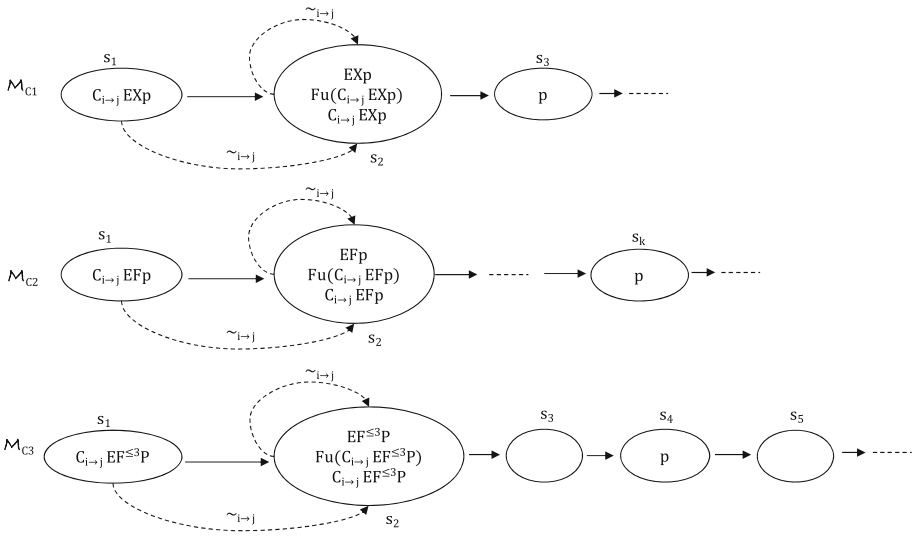

Fig. 3 Illustration of the semantics of commitment and its fulfillment

Example 1 Let us assume the models depicted in the Fig. 3. The state s_1 in $\mathcal{M}_{C1}$ is labeled with the commitment from i to j to bring about EXp because: (1) there is only one accessible state s_2; and (2) the formula EXp is true at s_2. The other models are the same but with different commitment contents. According to the semantics of fulfillment, the state s_2 in $\mathcal{M}_{C1}$ is also labeled with the fulfillment of the commitment because: (1) s_2 is accessible from s_1; and (2) such a commitment has been established at s_1.

Furthermore, our logic does not include an additional operator for violation as in [5,21]; instead weak and strong violations can be expressed as follows:

$$\neg AG(C_{i\to j}\varphi \;\rightarrow\; AF(Fu(C_{i\to j}\varphi))) \equiv EF(C_{i\to j}\varphi \wedge EG(\neg Fu(C_{i\to j}\varphi)))$$

and

$$\neg AG(C_{i\to j}\varphi \;\rightarrow\; EF(Fu(C_{i\to j}\varphi))) \equiv EF(C_{i\to j}\varphi \wedge AG(\neg Fu(C_{i\to j}\varphi)))$$

The weak violation takes place when there is a computation so that in its future a commitment is established but from the moment where the commitment is active there is a possible computation where globally the fulfillment never happens. The strong violation comes out when after having the commitment, the fulfillment does not occur in all states of every possible computation. The following proposition is a direct result from the semantics.

Proposition 1 *When the commitment is fulfilled, then there is no way to violate it in the future and vice versa.*

Lemma 1 *The social accessibility relation $\sim_{i\to j}$ is serial, transitive, and Euclidean.*

Proof

– $\sim_{i\to j}$ is serial: this follows from the assumption that for any pair $i, j \in \mathcal{A}$, we have that for any $s \in S$, $\sim_{i\to j}(s) \neq \emptyset$.

- $\sim_{i\to j}$ is transitive: assume $s \sim_{i\to j} s'$ and $s' \sim_{i\to j} s''$, for any pair $i, j \in \mathcal{A}$, according to the definition of $\sim_{i\to j}$, it is the case that $s \sim_{i\to j} s''$ as $l_i(s) = l_i(s') = l_i(s''), Var_i \cap Var_j \neq \emptyset, l_i^x(s) = l_i^x(s') = l_j^x(s'')\ \forall x \in Var_i \cap Var_j$, and $l_j^y(s) = l_j^y(s') = l_j^y(s'')\ \forall y \in Var_j - Var_i$.
- $\sim_{i\to j}$ is Euclidean: assume $s \sim_{i\to j} s'$ and $s \sim_{i\to j} s''$, for any pair $i, j \in \mathcal{A}$, according to the definition of $\sim_{i\to j}$, we have $s' \sim_{i\to j} s''$ as $l_i(s') = l_i(s) = l_i(s''), Var_i \cap Var_j \neq \emptyset, l_i^x(s') = l_i^x(s) = l_j^x(s'')\ \forall x \in Var_i \cap Var_j$, and $l_j^y(s') = l_j^y(s) = l_j^y(s'')\ \forall y \in Var_j - Var_i$. □

According to this observation, we can immediately conclude that the logic of commitments that deals with agent communication via social commitments is at least as strong as $KD45n$ (where n is the number of agents) which is to be expected. From Lemma 1, we obtain the following straightforward corollary.

Corollary 1 *The social accessibility relation $\sim_{i\to j}$ is shift reflexive, i.e. if $s \sim_{i\to j} s'$ then $s' \sim_{i\to j} s'$.*

Proposition 2 *The following validity holds:*

$$\models Fu(C_{i\to j}\varphi) \supset C_{i\to j}\varphi$$

Proof From the semantics of fulfillment in a state s, there is a state s' satisfying the commitment $C_{i\to j}\varphi$ and from which s is accessible. As $\sim_{i\to j}$ is transitive (Lemma 1), all the states accessible from s are also accessible from s'. From the semantics of $C_{i\to j}\varphi$, all those accessible states from s' and thus from s satisfy φ; so the result. □

Proposition 2 says that if the commitment is fulfilled, the commitment appears in the same state as its fulfillment. It is worth noticing that the property expressed in this proposition is not commonly accepted in the literature about the semantics of commitment actions, for example [50,59,66], according to which when a commitment is fulfilled, the commitment does not hold anymore in the same state where its content holds. Nevertheless, for the purpose of the reduction-based model checking we propose in this paper, this property is needed and it allows us to have the following proposition (Proposition 3) saying "the commitment should be active when it comes time to its fulfillment", which is very important as it implies the impossibility of fulfilling a nonexisting commitment (see Fig. 4).

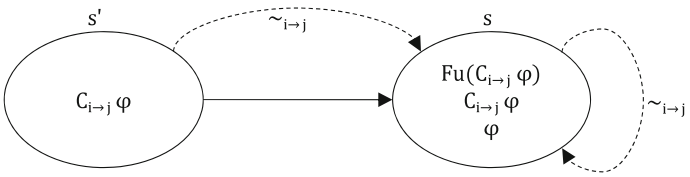

Fig. 4 Link between commitment and its fulfillment. s' is the state of activating the commitment and s is the state of fulfilling the commitment. Notice that the commitment $C_{i\to j}\varphi$ activated in s' is still active in s as the accessibility relation is shift reflexive

Proposition 3 *The following validity holds:*

$$\models AG(\neg E(\neg C_{i\to j}\varphi\ U\ (\neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi))))$$

Proof Let us assume the opposite is true, which means: $EF(E(\neg C_{i\to j}\varphi\ U\ (\neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi))))$. According to the semantics of until, $\neg C_{i\to j}\varphi\ U\ (\neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi))$ is satisfied in a path if it has $\neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi)$ in its future. However, from Proposition 2, this cannot be the case; so the result. □

As our objective in this paper is to investigate the practical problem of model checking commitments for communicating agents, the completeness issue will not be considered, so that the paper is more focussed on the implementation part of the verification problem and its computational complexity.

3 Model checking CTLC$^+$ using reduction

Model checking is a formal and automatic technique used to verify finite state concurrent systems. It was independently developed by Clarke and Emerson, and by Sifakis and Queille in the early of 1980s. Their work over the years has led to the creation of new logics for specification, new verification algorithms and techniques, and new model checking tools which are available today. Verifying MASs has become an important subfield on its own. In this section, we proceed to present our reduction techniques to model checking (the second part in our approach). In these techniques, we reduce the problem of model checking CTLC$^+$ into the problem of model checking ARCTL and into the problem of model checking GCTL*. Before that, we define the problem of model checking CTLC$^+$: in a nutshell, given a MAS represented as an interpreted system $\mathcal{M}_C$ and a formula φ in CTLC$^+$ describing a property, the problem of model checking CTLC$^+$ can be defined as establishing whether or not $\mathcal{M}_C \models \varphi$, i.e., $\forall s \in I : (\mathcal{M}_C, s) \models \varphi$.

3.1 Reducing CTLC$^+$ into ARCTL

We briefly review ARCTL logic (an extension of CTL with action formulae) [42]. We then show how the problem of model checking CTLC$^+$ can be reduced to the problem of ARCTL model checking. ARCTL logic mixes state formulae and action formulae by restricting path formulae to paths whose actions satisfy a given action formula. The syntax of ARCTL is defined by the following BNF grammar [42]:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid E_\alpha X\varphi \mid E_\alpha(\varphi\ U\ \varphi) \mid E_\alpha G\varphi$$
$$\alpha ::= b \mid \neg\alpha \mid \alpha \vee \alpha$$

where φ is a state formula, α is an action formula, $p \in \Phi_p$ is an atomic proposition, and $b \in \Phi_b$ is an atomic action proposition.

Definition 5 (*Model of ARCTL*) A model $\mathcal{M}_A = (S_A, I_A, Act, TR, V_S, V_{Act})$ is a tuple where S_A is a nonempty set of states; $I_A \subseteq S_A$ is a set of initial states; Act is a set of actions; $TR \subseteq S_A \times Act \times S_A$ is a labeled transition relation; $V_S : S_A \to 2^{\Phi_P}$ is a function assigning to each state a set of atomic propositions to interpret this state; and $V_{Act} : Act \to 2^{\Phi_b}$ is a function assigning to each action a set of atomic action propositions to interpret this action.

The semantics of this logic [42] is given by defining the α-restriction of $\mathcal{M}_A = (S_A, I_A, Act, TR, V_S, V_{Act})$ as follows $\mathcal{M}_A^\alpha = (S_A, I_A, Act, TR^\alpha, V_S, V_{Act})$ where TR^α

is a transition relation such that $(s, a, s') \in TR^{\alpha}$ iff $(s, a, s') \in TR$ and $a \models \alpha$ wherein $\models$ is defined as follows:

- $a \models b$ iff $b \in V_{Act}(a)$;
- $a \models \neg\alpha$ iff not $(a \models \alpha)$ and;
- $a \models \alpha \vee \alpha'$ iff $a \models \alpha$ or $a \models \alpha'$.

The motivation behind the α-restriction is to focus each time on specific transitions whose labels satisfy a given action formula, so all the other transitions are disregarded. This is useful when a formula has to be checked because only relevant transitions should be considered.

Pecheur and Raimondi [42] have considered finite and infinite paths to define the semantics of ARCTL. However, we only consider the general case of infinite paths. $\Pi^{\alpha}(s)$ is the set of paths (called α-paths) whose actions satisfy a given action formula α and starting at s. Now, we define the satisfaction relation $(\mathcal{M}_A^{\alpha}, s) \models \varphi$, or concisely $s \models \varphi$, as follows (we omit the semantics of Boolean connectives and propositional atoms):

$s \models E_{\alpha} X\varphi$ iff there exists a path $\pi \in \Pi^{\alpha}(s)$ and $\pi(1) \models \varphi$,

$s \models E_{\alpha}(\varphi\ U\ \psi)$ iff there exists a path $\pi \in \Pi^{\alpha}(s)$ such that for some $k \geq 0, \pi(k) \models \psi$ and $\pi(j) \models \varphi$ for all $0 \leq j \leq k-1$,

$s \models E_{\alpha} G\varphi$ iff there exists a path $\pi \in \Pi^{\alpha}(s)$ such that $\pi(k) \models \varphi$ for all $k \geq 0$.

The reduction process is defined as follows: given a CTLC$^+$ model $\mathcal{M}_C = (S, I, R_t, \{\sim_{i\to j} | (i, j) \in \mathcal{A}^2\}, \mathcal{V})$ and a CTLC$^+$ formula φ, we have to define an ARCTL model $\mathcal{M}_A^{\alpha} = \mathscr{F}(\mathcal{M}_C)$ and an ARCTL formula $\mathscr{F}(\varphi)$ using a transformation function $\mathscr{F}$ such that $\mathcal{M}_C \models \varphi$ iff $\mathscr{F}(\mathcal{M}_C) \models \mathscr{F}(\varphi)$. The model $\mathscr{F}(\mathcal{M}_C)$ is defined as an ARCTL model $\mathcal{M}_A^{\alpha} = (S_A, I_A, Act, TR^{\alpha}, V_S, V_{Act})$ as follows:

- $S_A = S$; $I_A = I$; $V_S = \mathcal{V}$,
- the set Φ_b is defined as follows: $\Phi_b = \{\epsilon, \alpha_{1\to 1}, \alpha_{1\to 2}, \ldots, \alpha_{n\to n}\} \cup \{\beta_{1\to 1}, \beta_{1\to 2}, \ldots, \beta_{n\to n}\}$, and then $Act = \{\alpha^o, \alpha^{11}, \alpha^{12}, \ldots, \alpha^{nn}\} \cup \{\beta^{11}, \beta^{12}, \ldots, \beta^{nn}\}$ where α^o and α^{ij} are the actions labeling transitions respectively defined from the transition relation R_t and the accessibility relation $\sim_{i\to j}$, while β^{ij} is the action labeling transitions added when there exists a transition labeled with α^{ij} and needed to define transformation of the formula $Fu(C_{i\to j}\varphi)$,
- the function V_{Act} is then defined as follows:
  1. $V_{Act}(\alpha^o) = \{\epsilon\}$, i.e., ϵ is the atomic action proposition forming α^o,
  2. $V_{Act}(\alpha^{ij}) = \{\alpha_{i\to j}\}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$, i.e., $\alpha_{i\to j}$ is the atomic action proposition forming α^{ij},
  3. $V_{Act}(\beta^{ij}) = \{\beta_{i\to j}\}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$, i.e., $\beta_{i\to j}$ is the atomic action proposition forming β^{ij}.
- the labeled transition relation TR^{α} combines the temporal labeled transition R_t and the accessibility relation $\sim_{i\to j}$ under the following conditions: for states $s, s' \in S$,
  1. $(s, \alpha^0, s') \in TR^{\epsilon}$ if $(s, s') \in R_t$,
  2. $(s, \alpha^{ij}, s') \in TR^{\alpha_{i\to j}}$ if $s \sim_{i\to j} s'$,
  3. $(s, \beta^{ij}, s') \in TR^{\beta_{i\to j}}$ if $(s', \alpha^{ij}, s) \in TR^{\alpha_{i\to j}}$.

From the definition of $\mathscr{F}(\mathcal{M}_C)$, it is clear that $\mathscr{F}(S) = S_A$. Let us now define the transformation of a CTLC$^+$ formula φ (i.e., $\mathscr{F}(\varphi)$) by induction on the form of φ.

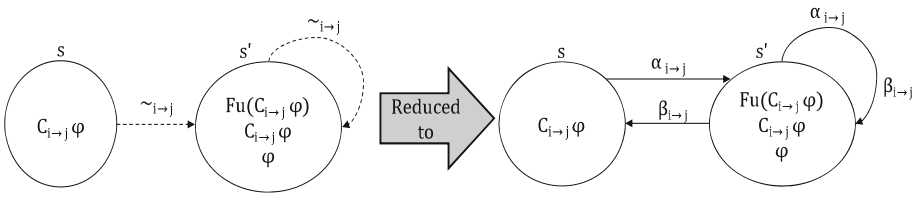

Fig. 5 An example of the reduction process from CTLC$^+$ to ARCTL

- $\mathscr{F}(p) = p$, if p is an atomic proposition,
- $\mathscr{F}(\neg\varphi) = \neg\mathscr{F}(\varphi)$, and $\mathscr{F}(\varphi \vee \psi) = \mathscr{F}(\varphi) \vee \mathscr{F}(\psi)$,
- $\mathscr{F}(EX\varphi) = E_\epsilon X\mathscr{F}(\varphi)$, and $\mathscr{F}(E(\varphi\ U\ \psi)) = E_\epsilon(\mathscr{F}(\varphi)\ U\ \mathscr{F}(\psi))$,
- $\mathscr{F}(EG\varphi) = E_\epsilon G\mathscr{F}(\varphi)$, and $\mathscr{F}(C_{i\to j}\varphi) = A_{\alpha_{i\to j}} X\mathscr{F}(\varphi)$,
- $\mathscr{F}(Fu(C_{i\to j}\varphi)) = E_{\beta_{i\to j}} X\mathscr{F}(C_{i\to j}\varphi)$.

Thus, this reduction allows us to model check CTLC$^+$ formulae by model checking their reductions in ARCTL using the extended NuSMV tool introduced in [35]. Figure 5 illustrates the reduction process of the fulfillment formula. The following theorem proves the soundness of our reduction from CTLC$^+$ to ARCTL.

Theorem 1 (Soundness of $\mathscr{F}$) *Let $\mathcal{M}_C$ and φ be respectively a CTLC$^+$ model and formula and let $\mathscr{F}(\mathcal{M}_C)$ and $\mathscr{F}(\varphi)$ be the corresponding model and formula in ARCTL. We have $\mathcal{M}_C \models \varphi$ iff $\mathscr{F}(\mathcal{M}_C) \models \mathscr{F}(\varphi)$.*

Proof We prove this theorem by induction on the structure of the formula φ. All the cases are straightforward once the following two cases are analyzed:

- $\varphi = C_{i\to j}\psi$. We have $(\mathcal{M}_C, s) \models C_{i\to j}\psi$ iff $(\mathcal{M}_C, s') \models \psi$ for every $s' \in S$ such that $s \sim_{i\to j} s'$. Consequently, $(\mathcal{M}_C, s) \models C_{i\to j}\psi$ iff $(\mathcal{M}_A^{\alpha_{i\to j}}, s') \models \mathscr{F}(\psi)$ for every $s' \in S_A$ such that $(s, \alpha^{ij}, s') \in TR^{\alpha_{i\to j}}$. As $\sim_{i\to j}$ is shift reflexive, we obtain an infinite path $\pi \in \Pi^{\alpha_{i\to j}}(s)$ such that $\pi(1) = s'$ and $(\mathcal{M}_A^{\alpha_{i\to j}}, \pi(1)) \models \mathscr{F}(\psi)$ (see Fig. 5). By semantics of $A_{\alpha_{i\to j}} X$ in ARCTL, we obtain $(\mathcal{M}_A^{\alpha_{i\to j}}, s) \models A_{\alpha_{i\to j}} X\mathscr{F}(\psi)$.
- $\varphi = Fu(C_{i\to j}\psi)$. We have $(\mathcal{M}_C, s') \models Fu(C_{i\to j}\psi)$ iff $(\mathcal{M}_C, s) \models C_{i\to j}\psi$ for a state $s \in S$ such that $s \sim_{i\to j} s'$. Consequently, $(\mathcal{M}_C, s') \models Fu(C_{i\to j}\psi)$ iff $(\mathcal{M}_A^{\alpha_{i\to j}}, s) \models \mathscr{F}(C_{i\to j}\psi)$ for a state $s \in S_A$ such that $(s, \alpha^{ij}, s') \in TR^{\alpha_{i\to j}}$. As $\sim_{i\to j}$ is shift reflexive, we obtain $s' \sim_{i\to j} s'$ and so $(s', \alpha^{ij}, s') \in TR^{\alpha_{i\to j}}$. Consequently, $(s', \beta^{ij}, s') \in TR^{\beta_{i\to j}}$. There is then an infinite path $\pi \in \Pi^{\beta_{i\to j}}(s')$ such that $\pi(1) = s'$ and $(\mathcal{M}_A^{\beta_{i\to j}}, \pi(1)) \models \mathscr{F}(C_{i\to j}\psi)$ (see Fig. 5). By semantics of $E_{\beta_{i\to j}} X$ in ARCTL, we obtain $(\mathcal{M}_A^{\beta_{i\to j}}, s') \models E_{\beta_{i\to j}} X\ \mathscr{F}(C_{i\to j}\psi)$, so we are done. □

3.2 Reducing CTLC$^+$ into GCTL*

As in the reduction of CTLC$^+$ into ARCTL, we start with briefly reviewing Generalized CTL* (GCTL*), a logic that extends CTL* by allowing formulae to constrain actions as well as states [10]. We then show how the problem of model checking CTLC$^+$ can be reduced to the problem of model checking GCTL*. The syntax of GCTL* is defined by the following BNF grammar [10]:

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

where $p \in \Phi_p$, Φ_p is a set of atomic propositions and $\theta \in \Phi_b$, Φ_b is a set of atomic action propositions. In this syntax, there are two kind of formulae: state formulae $\mathcal{S}$ and path formulae $\mathcal{P}$. 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*.

Definition 6 (*Model of* GCTL*) A model $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ is a tuple where S_G is a nonempty set of states; Ac is a set of actions; $l_S : S_G \rightarrow 2^{\Phi_p}$ is a state labeling function; $l_{Ac} : Ac \rightarrow 2^{\Phi_b}$ 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 follows a standard convention in temporal logic, such as CTL* [10]. 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. X 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 CTLC$^+$ to the problem of model checking GCTL* that allows a direct use of CWB-NC is defined as follows: given a CTLC$^+$ model $\mathcal{M}_C = (S, I, R_t, \{\sim_{i\rightarrow j} \mid (i, j) \in \mathcal{A}^2\}, \mathcal{V})$ and a CTLC$^+$ formula φ, we have to define a GCTL* model $\mathcal{M}_G = \mathscr{H}(\mathcal{M}_C)$ and a GCTL* formula $\mathscr{H}(\varphi)$ using a transformation function $\mathscr{H}$ such that $\mathcal{M}_C \models \varphi$ iff $\mathscr{H}(\mathcal{M}_C) \models \mathscr{H}(\varphi)$. The model $\mathscr{H}(\mathcal{M}_C)$ is defined as a GCTL* model $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ as follows:

- $S_G = S$; $I_G = I$; $l_S = \mathcal{V}$,
- to define the set Ac, let us first define the set of atomic action propositions $\Phi_b = \{\epsilon, \alpha_{1\rightarrow 1}, \alpha_{1\rightarrow 2}, \ldots, \alpha_{n\rightarrow n}\} \cup \{\beta_{1\rightarrow 1}, \beta_{1\rightarrow 2}, \ldots, \beta_{n\rightarrow n}\}$, then $Ac = \{\alpha^o, \alpha^{11}, \alpha^{12}, \ldots, \alpha^{nn}\} \cup \{\beta^{11}, \beta^{12}, \ldots, \beta^{nn}\}$ where α^o and α^{ij} are the actions labeling transitions respectively defined from the transition relation R_t and the accessibility relation $\sim_{i\rightarrow j}$, while β^{ij} is the action labeling transitions added when there exists a transition labeled with α^{ij} and needed to define transformation of the formula $Fu(C_{i\rightarrow j}\varphi)$,
- the function l_{Ac} is then defined as follows:

  1. $\alpha^o \in Ac$, then $l_{Ac}(\alpha^o) = \{\epsilon\}$,
  2. $l_{Ac}(\alpha^{ij}) = \{\alpha_{i\rightarrow j}\}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$,
  3. $l_{Ac}(\beta^{ij}) = \{\beta_{i\rightarrow j}\}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$.

- the labeled transition relation $\rightarrow$ combines the temporal labeled transition R_t and the accessibility relation $\sim_{i\rightarrow j}$ under the following conditions: for states $s, s' \in S$,

  1. $(s, \alpha^o, s') \in \rightarrow$ if $(s, s') \in R_t$,
  2. $(s, \alpha^{ij}, s') \in \rightarrow$ if $s \sim_{i\rightarrow j} s'$,
  3. $(s, \beta^{ij}, s') \in \rightarrow$ if $(s', \alpha^{ij}, s) \in \rightarrow$.

From the definition of $\mathscr{H}(\mathcal{M}_C)$, it is clear that $\mathscr{H}(S) = S_G$. Let us now define $\mathscr{H}(\varphi)$ by induction on the form of φ.

- $\mathscr{H}(p)$=p, if p is an atomic proposition,
- $\mathscr{H}(\neg\varphi) = \neg\mathscr{H}(\varphi)$, and $\mathscr{H}(\varphi \vee \psi) = \mathscr{H}(\varphi) \vee \mathscr{H}(\psi)$,

- $\mathscr{H}(EX\varphi) = EX\mathscr{H}(\varphi)$, and $\mathscr{H}(E(\varphi\ U\ \psi)) = E(\mathscr{H}(\varphi)\ U\ \mathscr{H}(\psi))$,
- $\mathscr{H}(EG\varphi) = EG\mathscr{H}(\varphi)$, and $\mathscr{H}(C_{i\to j}\varphi) = A(\alpha_{i\to j} \to X\mathscr{H}(\varphi))$,
- $\mathscr{H}(Fu(C_{i\to j}\varphi)) = E(\beta_{i\to j} \wedge X\mathscr{H}(C_{i\to j}\varphi))$.

Theorem 2 (Soundness of $\mathscr{H}$) *Let $\mathcal{M}_C$ and φ be respectively a CTLC$^+$ model and formula and let $\mathscr{H}(\mathcal{M}_C)$ and $\mathscr{H}(\varphi)$ be the corresponding model and formula in GCTL*. We have $\mathcal{M}_C \models \varphi$ iff $\mathscr{H}(\mathcal{M}_C) \models \mathscr{H}(\varphi)$.*

Proof We can prove this theorem by induction on the structure of φ in a way similar to the proof of Theorem 1. □

Remark 1 GCTL* and ARCTL logics can both express properties of state-based and action-based models and a careful analysis of the semantics of these two logics reveals that GCTL* subsumes ARCTL. In fact, in GCTL*, a path satisfies θ iff the path contains at least one transition and the label of the first transition on the path satisfies θ where θ is an atomic action proposition, and an action a satisfies θ iff $\theta \in l_{Ac}(a)$. In ARCTL, the actions of the whole path, not only of the first transition, should satisfy an action formula α as the transitions in the model of ARCTL are α-restricted. In fact, ARCTL allows quantification over action-labelled paths, so a path formula is evaluated on α-paths, which means paths where all transitions are α-restricted, which can also be done in GCTL*. For example, for a path formula γ, $E_\alpha\gamma$ in ARCTL means there is an α-path where γ holds, which can be expressed in GCTL* as: $E(G\alpha \wedge \gamma)$. However, although GCTL* subsumes ARCTL, their model checking techniques are different and our motivation behind using these two logics is to be able to use their respective model checkers, which are based on two different model checking techniques: automata-based technique for GCTL* and symbolic, OBDD-based technique for ARCTL.

3.3 Complexity analysis

Overview

In the previous sections, explicit models ($\mathcal{M}_C$, $\mathcal{M}_A$, and $\mathcal{M}_G$) were considered. In this section, we analyze the space complexity of CTLC$^+$ model checking for explicit models and then its complexity for concurrent programs. We use concurrent programs as defined in [34] as composed of n concurrent processes (agents) P_i, where each process is described by a transition system (the formal definition is given later). The need for concurrent programs is motivated by the need of having compact representations where states and transitions are not listed explicitly, but having instead compact representations that still correspond to the actual system. In general, the relation between explicit models (i.e. Kripke-like structures) and concurrent programs, which provide compact representations of the systems to be checked, is as stated in [34]: "the Kripke structures to which model checking is applied are often obtained by constructing the *reachability graph of concurrent programs*". In other terms, the explicit models are obtained as the product of the components P_i of concurrent programs. The size of explicit models is thus exponential in the size of processes P_i as the system's evolution results from joint actions of the components [29].

To analyze the complexity of model checking CTLC$^+$ in concurrent programs, we use a methodology similar to the one presented in [34]. The idea is as follows. First, we analyze the complexity of model checking GCTL* in the explicit model $\mathcal{M}_G$, and show that by using an on-the-fly (local) and top–down algorithm, it is possible to perform model checking GCTL* in space *polynomial* in the length of the formula, but only *poly-logarithmic* in the size of the explicit model. It is important to mention that the algorithm is on-the-fly, which means we do

not hold the whole structure to be checked in memory at any one time, and this is the reason behind the poly-logarithmic space complexity in the size of the explicit model. As in [34], our approach is an automata-theoretic approach, and makes use of a special class of automata called ABTA [4,10], which will be introduced later. The approach is based on building an ABTA, combining the model $\mathcal{M}_G$ and the automaton of the formula to be verified, and checking its nonemptiness. This combined ABTA is computed on-the-fly and limited to its reachable states, which avoids exploring the parts of the model $\mathcal{M}_G$ that are irrelevant for the formula to be checked. The type of ABTA employed allows using a top–down, space-efficient model checking algorithm. Then, we prove that the explicit structure complexity of GCTL* model checking (i.e. by fixing the formula) is NLOGSPACE-complete, which means that model checking GCTL* is NLOGSPACE-complete in the size $|\mathcal{M}_G|$ of the explicit model. Thereafter, we use the previous results to obtain the complexity of model checking GCTL* for concurrent programs, exploiting the fact that the combined ABTA whose nonemptiness has to be checked is obtained as the product of the components of a concurrent program and this product is at most exponentially larger than the program itself. Thus, the fact that (1) the space complexity of model checking GCTL* is polynomial in the length of the formula and poly-logarithmic in the size of the explicit model; and (2) the model checking algorithm is on-the-fly, imply that GCTL* model checking for a concurrent program can be done in polynomial space with respect to the size of this program rather than of the order of the exponentially larger combined ABTA as is the case of bottom-up approaches to model checking. By logspace reduction to GCTL* model checking with respect to explicit models, we analyze the explicit structure complexity of CTLC$^+$ model checking and prove that is NLOGSPACE-complete, which, as the case of GCTL*, implies that CTLC$^+$ model checking can be done in polynomial space with respect to the size of concurrent programs.

Preliminaries

We start this section by defining ABTA and associated concepts needed to analyze the complexity of model checking GCTL*. Definition of concurrent programs will follow. We use $\mathcal{L} = \Phi_p \cup \{\neg p \mid p \in \Phi_p\}$ to denote the set of state literals and $\mathcal{L}_{act} = \Phi_b \cup \{\neg\theta \mid \theta \in \Phi_b\}$ to denote the set of action literals. Let Θ be a typical subset of $\mathcal{L}_{act}$. An ABTA is defined as follows [10]:

Definition 7 (*ABTA*) An ABTA $\mathcal{B}$ is a tuple $(Q, h, \rightarrow_B, q_I, \mathcal{F})$, where Q is a finite set of states; $h : Q \rightarrow \mathcal{L} \cup \{\neg, \wedge, \vee, [\Theta], \langle\Theta\rangle\}$ is the state labeling function; $\rightarrow_B \subseteq Q \times Q$ is the transition relation; $q_I \in Q$ is the start state; and $\mathcal{F} \subseteq 2^Q$ is the set of sets of accepting states. $\rightarrow_B$ should also satisfy:

$$|\{q' | q \rightarrow_B q'\}| \begin{cases} = 0 \text{ if } h(q) \in \mathcal{L} \\ = 1 \text{ if } h(q) \in \{\neg, [\Theta], \langle\Theta\rangle\} \\ \geq 1 \text{ if } h(q) \in \{\wedge, \vee\} \end{cases}$$

Also if $h(q) = \neg$, then q does not appear on a cycle.

ABTAs have the advantage of supporting efficient model checking for different logics and are used to define the system properties using tableau proof rules [9]. They are used to encode how the properties are to be proved and allow us to encode top–down proofs for temporal formulae (GCTL* formulae in this case). Indeed, an ABTA $\mathcal{B}$ encodes a proof schema in order to prove, in a goal-directed manner, that a model $\mathcal{M}_G$ satisfies a temporal formula. Let us consider the example of proving that a state s in a model $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$

satisfies a temporal formula of the form $F_1 \wedge F_2$, where F_1 and F_2 are two formulae. Regardless of the structure of the system, there would be two subgoals if we want to prove this in a top–down, goal-directed manner. The first would be to prove that s satisfies F_1, and the second would be to prove that s satisfies F_2. Intuitively, an ABTA for $F_1 \wedge F_2$ would encode this proof structure using states for the formulae $F_1 \wedge F_2$, F_1, and F_2. A transition from $F_1 \wedge F_2$ to each of F_1 and F_2 should be added to the ABTA and the labeling of the state for $F_1 \wedge F_2$ being "$\wedge$". Indeed, in an ABTA, we can consider that: (1) states correspond to "formulae"; (2) the labeling of a state is the "logical operator" used to construct the formula or a state literal from $\mathcal{L}$; and (3) the transition relation represents a "subgoal" relationship. Thus, to show that a model state s satisfies an ABTA state q labeled with $\wedge$, one needs to show that s satisfies each of q's children. Regarding the labels $[\Theta]$ and $\langle\Theta\rangle$, for a model state s to satisfy an ABTA state q labeled with $[\Theta]$ (resp. $\langle\Theta\rangle$), one needs to show that for each s' (resp. some s') such that $(s, \alpha, s') \in\!\!\rightarrow$ for some α "satisfying" Θ (i.e. $\theta \in l_{Ac}(\alpha)$ for every $\theta \in \Theta$), s' must satisfy the unique successor of q.

In order to decide about the satisfaction of formulae, the notion of accepting runs of an ABTA $\mathcal{B}$ on a model $\mathcal{M}_G$ is used. These runs are infinite and cycle infinitely many times through accepting states. Formally, a run is defined as follows, where the notation $(s, \Theta, s') \in\!\!\rightarrow$ means $(s, \alpha, s') \in\!\!\rightarrow$ for some $\alpha \in Ac$ satisfying Θ:

Definition 8 (*Run of ABTA*) A run of an ABTA $\mathcal{B} = (Q, h, \rightarrow_B, q_I, \mathcal{F})$ on a model $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ is a maximal tree in which the nodes are classified as positive or negative and are labeled by elements of $Q \times S_G$ as follows.

- The root of the tree is a positive node and is labeled with (q_I, i_G) where $i_G \in I_G$
- If σ is a positive (resp. negative) node with label (q, s) such that $h(q) = \neg$ and $q \rightarrow_B q'$, then σ has one negative (resp. positive) child labeled (q', s)
- For a positive node σ labeled with (q, s):
  - If $h(q) \in \mathcal{L}$ then σ is a leaf.
  - If $h(q) = \wedge$ and $\{q' |\ q \rightarrow_B q'\} = \{q_1, \ldots, q_m\}$, then σ has m positive children labeled by (q_i, s), $1 \leq i \leq m$.
  - If $h(q) = \vee$, then σ has one positive child[4] labeled by (q', s) for some $q' \in \{q' |\ q \rightarrow_B q'\}$.
  - If $h(q) = [\Theta]$, q' is such that $q \rightarrow_B q'$, and $\{s' |\ (s, \Theta, s') \in\!\!\rightarrow\} = \{s_1, \ldots, s_m\}$, then σ has m positive children labeled by (q', s_i), $1 \leq i \leq m$.
  - If $h(q) = \langle\Theta\rangle$ and q' is such that $q \rightarrow_B q'$, then σ has one positive child labeled by (q', s') for some s' such that $(s, \Theta, s') \in\!\!\rightarrow$.
- Otherwise, for a negative node σ labeled with (q, s):
  - If $h(q) \in \mathcal{L}$ then σ is a leaf.
  - If $h(q) = \wedge$, then σ has one negative child[5] labeled by (q', s) for some $q' \in \{q' |\ q \rightarrow_B q'\}$.
  - If $h(q) = \vee$ and $\{q' |\ q \rightarrow_B q'\} = \{q_1, \ldots, q_m\}$, then σ has m negative children labeled by (q_i, s), $1 \leq i \leq m$.
  - If $h(q) = [\Theta]$ and q' is such that $q \rightarrow_B q'$, then σ has one negative child labeled by (q', s') for some s' such that $(s, \Theta, s') \in\!\!\rightarrow$.

[4] We only consider one positive child in a run when the node is positive and disjunctive (i.e., labeled by $\vee$ or $\langle\Theta\rangle$) as only one branch in the product graph (see Definition 11) is chosen.

[5] Similar to the positive case, we only consider one negative child in a run when the node is negative and conjunctive (i.e., labeled by $\wedge$ or $[\Theta]$) because again only one branch in the product graph is selected.

– If $h(q) = \langle\Theta\rangle$, q' is such that $q \rightarrow_B q'$, and $\{s' |\ (s, \Theta, s') \in \rightarrow\} = \{s_1, \ldots, s_m\}$, then σ has m negative children labeled by (q', s_i), $1 \leq i \leq m$.

Every infinite path in a well-formed ABTA has a suffix that contains either only positive or only negative nodes [10]. If only positive (resp. negative) nodes are included, the path is said to be positive (resp. negative). A successful run is then defined as follows:

Definition 9 (*Successful run of ABTA*) Let R be a run of an ABTA $\mathcal{B}$ on a model $\mathcal{M}_G$.

– A positive leaf of R labeled (q, s) is successful iff s satisfies $h(q)$ or $h(q) = [\Theta]$ and there is no s' such that $(s, \Theta, s') \in \rightarrow$.
– A negative leaf of R labeled (q, s) is successful iff s does not satisfy $h(q)$ or $h(q) = \langle\Theta\rangle$ and there is no s' such that $(s, \Theta, s') \in \rightarrow$.
– A positive path is successful iff for each $\mathbf{F} \in \mathcal{F}$ some $q \in \mathbf{F}$ occurs infinitely often.
– A negative path is successful iff for some $\mathbf{F} \in \mathcal{F}$ there is no $q \in \mathbf{F}$ that occurs infinitely often.

R is successful iff every leaf and every path in R is successful.

Model checking GCTL*. Let ψ be a GCTL* state formula. The automata-based model checking procedure for GCTL* proposed in [10] works as follows:

1. Translating ψ into a variant of ABTA: and-restricted alternating Büchi tableau automaton (arABTA). The resulting automaton is denoted by $\mathcal{B}_\psi$.
2. Exploring the product graph of $\mathcal{M}_G$ and $\mathcal{B}_\psi$ to check if it contains a successful run. If such a run does exist, the formula is satisfied and $\mathcal{M}_G$ is said to be accepted by $\mathcal{B}_\psi$ (i.e. $\mathcal{M}_G \models \mathcal{B}_\psi$), otherwise, the formula is not satisfied. The product graph is denoted by $\mathcal{B}_{\mathcal{M}_G,\psi}$.

Definition 10 (*arABTA*) An ABTA $\mathcal{B}$ is and-restricted (arABTA) iff every state $q \in Q$ satisfies:

– If $h(q) = \wedge$ then there is at most one q' such that $q \rightarrow_B q'$ and there is a path from q' back to q.
– If $h(q) = [\Theta]$ and $q \rightarrow_B q'$, then there is no path from q' back to q.

In an arABTA, the strongly-connected component of a state labeled by $\wedge$ can contain ***at most*** one of its state's children and a state labeled by $[\Theta]$ is guaranteed to belong to a different strongly-connected component that its child. Thanks to this restrictedness, the handling of recursive children (i.e., children where there is a path from them back to the parents) is simplified, particularly when space is concerned. This makes simple the treatment of recursive calls needed for some GCTL* formulae, which allows for space-efficient model checking this logic (this will be made clear later when we will analyze the complexity of model checking GCTL*). As argued in [10], arABTAs play the same role in model checking GCTL* that do *Hesitant Alternating Word Automata* (HAAs) in model checking CTL and CTL* [34] although the two automata are conceptually different.[6] In fact, it has been shown that HAAs are the key to the space-efficient model checking algorithms for CTL and CTL* thanks to their restricted alternation structure (every nontrivial[7] strongly-connected component of HAAs is

[6] arABTA would be hesitant if for every strongly-connected component $Q_i \subseteq Q$ and every node $q \in Q_i$ either $h(q) \in \{\wedge, [\Theta]\}$ or $h(q) \in \{\vee, \langle\Theta\rangle\}$. Details about HAAs are out of scope of this paper and interested reader can refer to [34].

[7] A strongly-connected component Q_i is nontrivial if $|Q_i| > 1$ or $Q_i = \{q\}$ and q has a self loop.

either (1) existential, so contains only nodes that are disjunctively related; or (2) universal, so contains only nodes that are conjunctively related). In this paper, we will show that arABTAs are the key to the space-efficient complexity of the problem of model checking GCTL* and to the NLOGSPACE membership of the explicit structure complexity of this model checking problem.

Intuitively, the product graph of $\mathcal{M}_G$ and $\mathcal{B}_\psi$ can be seen as an encoding of all the runs of the arABTA. Formally:

Definition 11 (*Product graph*) The product graph $\mathcal{B}_{\mathcal{M}_G,\psi}$ of an arABTA $\mathcal{B}_\psi = (Q, h, \rightarrow_B, q_I, \mathcal{F})$ and a model $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ where $\mathcal{F} = \{F_0, \ldots, F_{n-1}\}$ has vertex set $Ver = Q \times S_G \times \{0, \ldots, n-1\}$ and edges $Edg \subseteq Ver \times Ver$. The edges are defined by: $((q, s, i), (q', s', i')) \in Edg$ iff

- there exist nodes σ and σ' in some run of $\mathcal{B}_\psi$ on $\mathcal{M}_G$ labeled (q, s) and (q', s') respectively such that σ' is a child of σ; and
- either $q \notin F_i$ and $i' = i$, or $q \in F_i$ and $i' = (i + 1) \mod n$

A vertex (q, s, i) is said to be accepting iff $q \in F$ for some $F \in \mathcal{F}$ and $i = 0$.

Bhat has proved in [9] that an arABTA can be partitioned uniquely to ordered sets $Q_1, \ldots, Q_n$, which correspond to its strongly-connected components. The number n of these sets is the depth of the arABTA. Finally, we define the sizes of $\mathcal{B}_\psi$, $\mathcal{M}_G$, and $\mathcal{B}_{\mathcal{M}_G,\psi}$ as follows:

- $|\mathcal{B}_\psi| = |Q| + |\mathcal{F}| + |\rightarrow_B|$ where $|Q|$ and $|\rightarrow_B|$ are the respective cardinalities of the sets Q and $\rightarrow_B$, and $|\mathcal{F}|$ is the number of component sets in $\mathcal{F}$.
- $|\mathcal{M}_G| = |S_G| + |Ac| + |\rightarrow|$.
- $|\mathcal{B}_{\mathcal{M}_G,\psi}| = |Ver| + |Edg|$, where the vertex set is bounded in size by $|Q| \cdot |S_G| \cdot |\mathcal{F}|$.

Concurrent programs. Let us now define concurrent programs. As introduced in [34], a concurrent program Pr is composed of n concurrent processes. Each process P_i is described by a transition system D_i defined as follows: $D_i = (AP_i, AC_i, S_i, \Delta_i, s_i^0, H_i)$ where AP_i is a set of local atomic propositions, AC_i is a local action alphabet, S_i is a finite set of local states, $\Delta_i \subseteq S_i \times AC_i \times S_i$ is a local transition relation, $s_i^0 \in S_i$ is an initial state, and $H_i : S_i \rightarrow 2^{AP_i}$ is a local state labeling function. A concurrent behavior of these processes is obtained by the product of the processes and transition actions that appear in several processes are synchronized by common actions. The joint behavior of the processes P_i can be described using a global transition system D, which is computed by constructing the reachable states of the product of the processes P_i and synchronization is obtained using common action names. Let $AP = \bigcup_{i=1}^n AP_i$, $AC = \bigcup_{i=1}^n AC_i$, $S = \Pi_{i=1}^n S_i$, $s^0 = (s_1^0, s_2^0, \ldots, s_n^0)$, $H(s) = \bigcup_{i=1}^n H_i(s[i])$ for every $s \in S$, and $s[i]$ be the ith component of s. Thus, $D = (AP, AC, S, \Delta, s^0, H)$ where $(s, a, s') \in \Delta$ iff $(s[i], a, s'[i]) \in \Delta_i$ or $s[i] = s'[i]$ for all $1 \leq i \leq n$.

*Complexity of model checking GCTL**

In this section, we prove two results: (1) the explicit structure complexity of GCTL* model checking (i.e. by fixing the formula) is NLOGSPACE-complete; and (2) model checking GCTL* for concurrent programs with respect to the size of the components P_i and the length of the formula being checked is PSPACE-complete.

Let $cl(\psi)$ be the closure of ψ defined as the smallest set such that the following hold:

- $\psi \in cl(\psi)$
- If $\neg\psi' \in cl(\psi)$ then $\psi' \in cl(\psi)$
- If $\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2 \in cl(\psi)$ then $\psi_1, \psi_2 \in cl(\psi)$
- If $E(\psi') \in cl(\psi)$ then $\psi' \in cl(\psi)$
- If $A(\psi') \in cl(\psi)$ then $E(\neg\psi') \in cl(\psi)$
- If $E(\psi_1 \wedge \psi_2) \in cl(\psi)$ then $E(\psi_1, \psi_2) \in cl(\psi)$
- If $E(\psi_1 \vee \psi_2) \in cl(\psi)$ then $E(\psi_1), E(\psi_2) \in cl(\psi)$
- If $E(\psi_1\ U\ \psi_2) \in cl(\psi)$ then $\psi_1, \psi_2, EX(\psi_1\ U\ \psi_2) \in cl(\psi)$
- If $EX\psi' \in cl(\psi)$ then $\psi' \in cl(\psi)$

We start by presenting the following proposition from [9] and [4], where $|\psi|$ denotes the length of the formula ψ measured as the number of elements in $cl(\psi)$.

Proposition 4 *Let ψ be a GCTL* state formula and $|\psi|$ be its length.*

1. $|\mathcal{B}_{\mathcal{M}_G,\psi}| = O(|\mathcal{M}_G| \cdot |\mathcal{B}_\psi|)$
2. $|\mathcal{B}_\psi| = O(2^{|\psi|})$

Theorem 3 *Given a GCTL* formula ψ, we can construct an arABTA $\mathcal{B}_\psi$ of size $O(2^{|\psi|})$ and of depth $O(|\psi|)$ such that every state in I_G, the set of initial states of $\mathcal{M}_G$, satisfies ψ iff $\mathcal{M}_G \models \mathcal{B}_\psi$.*

Proof Assuming that $\mathcal{B}_\psi$ is an arABTA, which will be shown later in this proof, the first part of the theorem is simply from Proposition 4. For the second part, the algorithm to generate arABTAs from GCTL* formulae uses goal-directed rules aiming to build tableaux from formulae. The formulae have the form $E(\Phi)$ and $A(\Phi)$ where Φ is a set of path formulae, so $E(\Phi)$ denotes $E(\bigwedge_{\varphi\in\Phi}\varphi)$, $A(\Phi)$ denotes $A(\bigvee_{\varphi\in\Phi}\varphi)$, and $E(\neg\Phi)$ denotes $E(\bigvee_{\varphi\in\Phi}\neg\varphi)$. Furthermore, we write $E(\Phi, \psi)$ to represent the formula of the form $E(\Phi\cup\{\psi\})$ and similarly $E(\Phi, \psi_1, \ldots, \psi_n)$ represents $E(\Phi \cup \{\psi_1\ \ldots, \psi_n\})$, where $\psi, \psi_1, \ldots \psi_n$ are path formulae and Φ is possibly empty. When Φ is empty, we can also write $E(\psi_1, \psi_2)$ to represent $E(\psi_1 \wedge \psi_2)$, which also represents $E(\{\psi_1\} \cup \{\psi_2\})$. To build an arABTA $\mathcal{B}_\psi$ from a state formula $\psi \equiv E(\Phi)$, one first generates the states and transitions. The initial state is the formula ψ itself and in general, states correspond to state formulae and transitions are linking formulae to their subformulae as defined by the closure of these formulae. The subformulae (in the sense of the closure) are obtained by applying the tableau rules shown in Table 1 in the order R1 to R10, where the top part of each rule being the goal and the bottom part being the subgoals. Assuming a state already associated with a formula ψ, one applies the rules R1–R10 to generate new states by comparing the form of ψ with the formula in the goal part of the rules starting from R1. When ψ and the goal formula match, the label of the rule becomes the label of the state and the subgoal formulae obtained from the rule are added as states and transitions from ψ to these states are added. Leaves are labeled by the state literals and the process stops when no new states are added. The soundness and termination of this algorithm are presented in [10].

Table 1 Tableau Rules for GCTL*

R1 $\wedge$: $\frac{\psi_1\wedge\psi_2}{\psi_1\,\psi_2}$	R2 $\vee$: $\frac{\psi_1\vee\psi_2}{\psi_1\mid\psi_2}$	R3 $\vee$: $\frac{E(\psi)}{\psi}$	R4 $\neg$: $\frac{\neg\psi}{\psi}$	R5 $\neg$: $\frac{A(\Phi)}{E(\neg\Phi)}$
R6 $\wedge$: $\frac{E(\Phi,\psi)}{E(\Phi)E(\psi)}$	R7 $\wedge$: $\frac{E(\Phi,\varphi_1\wedge\varphi_2)}{E(\Phi,\varphi_1,\varphi_2)}$	R8 $\vee$: $\frac{E(\Phi,\varphi_1\vee\varphi_2)}{E(\Phi,\varphi_1)E(\Phi,\varphi_2)}$		
R9 $\vee$: $\frac{E(\Phi,\varphi_1 U\varphi_2)}{E(\Phi,\varphi_2)E(\Phi,\varphi_1,X(\varphi_1 U\varphi_2))}$	R10 $\langle\Psi_1\rangle$: $\frac{E(\Psi,X\varphi_1,\ldots,X\varphi_n)}{E(\varphi_1,\ldots,\varphi_n)}$			

Ψ is an ordered set of action literals and Ψ_1 is a subset of Ψ containing only the first element of Ψ

Let us now show that the obtained automaton $\mathcal{B}_\psi$ is an arABTA. We start first by proving that $\mathcal{B}_\psi$ is an ABTA. From the tableau rules R1–R10 and the explanation above, we can see that states are labeled by a subset of $\{\neg, \wedge, \vee, [\Theta], \langle\Theta\rangle\}$, and: (1) leaves (states without children) are labeled by elements of $\mathcal{L}$; (2) states labeled by $\{\neg, \langle\Theta\rangle\}$ have only one child (rules R4, R5, and R10); (3) states labeled by $\{\wedge, \vee\}$ have at least one child (rules R1, R2, R3, R6, R7, R8, and R9); and (4) states labeled by $\neg$ using rules R4 and R5 do not appear on a cycle as there are no rules linking ψ to $\neg\psi$ or $E(\neg\Phi)$ to $A(\Phi)$ since $\neg\psi$ and $A(\Phi)$ do not appear as subgoal formulae in any of the rules. Consequently, $\mathcal{B}_\psi$ satisfies all the conditions of ABTAs (Definition 7), so it is an ABTA. To show that it is an arABTA, we only need to show that (1) states labeled by $[\Theta]$ have no recursive children; and (2) states labeled by $\wedge$ have at most one recursive child, which means in rules labeled by $\wedge$, at most one subgoal can be recursive, so it can include the formula identified in the goal. The first part is straightforward as no state is labeled by $[\Theta]$ in $\mathcal{B}_\psi$. For the second part, three rules should be discussed: R1, R6, and R7. R1 produces two children, but no one is recursive as there are no rules linking ψ_1 or ψ_2 back to $\psi_1 \wedge \psi_2$. R6 generates two children: $E(\psi)$ and $E(\Phi)$, but the child $E(\psi)$ is not recursive as the only available rule to be applied once $E(\psi)$ is obtained is R3, which will generate a state labeled by ψ and from ψ a formula having the form $E(\Phi, \psi)$ cannot be produced. Thus, R6 can produce at most one recursive child, which could be from $E(\Phi)$. Finally, R7 generates only one child, so again at most one is recursive.

The partition of the obtained arABTA $\mathcal{B}_\psi$ to $Q_1, \ldots, Q_n$ proceeds as follows: $q_I \in Q_n$ and for each state $q \in Q_i$ we have:

- if $h(q) \in \{\vee, \langle\Theta\rangle\}$ then for every q' such that $q \rightarrow_B q'$, $q' \in Q_j$ and $j \leq i$.
- if $h(q) \in \{[\Theta], \neg\}$ then for every q' such that $q \rightarrow_B q'$, $q' \in Q_j$ and $j < i$.
- if $h(q) = \wedge$ then there is exactly at most one state q' from the set $\{q' | \ q \rightarrow_B q'\}$ such that $q' \in Q_j$ and $j \leq i$. For the other states q' we have $q' \in Q_j$ and $j < i$.

Thus, since each state is associated to a subformula as defined in the closure, this partition shows that each subformula (in the sense of the closure) of a formula ψ induces at most one set Q_i in ψ. Therefore, the depth of $\mathcal{B}_\psi$ is linear in $|\psi|$. □

The following is an example from [9] showing the tableau and arABTA obtained from a given GCTL* formula.

Example 2 Let $Ac = \{send, receive\}$. Consider the formula $AG(send \rightarrow F(receive))$. The tableau of the formula along with the applied rules are shown in Table 2. The obtained arABTA is depicted in Fig. 6. It is worth noticing that in the first application of R9 (to obtain the formulae in row 3 of Table 2), we have in the goal part Φ is empty, $\varphi_1 \equiv true$, and $\varphi_2 \equiv send \wedge G(\neg receive)$. In the subgoal part, we use the form with "," instead of "$\wedge$" for the left side formula. This choice is simply motivated by the fact that the two components (i.e., $send$ and $G(\neg receive)$) become clearly separated, which allows this row (i.e., row 3) to match the goal part of rule R9. By so doing, it becomes clear that in the second application of R9 (to obtain the formula in row 4), $\Phi = \{send\}$ and $\varphi_1 U \varphi_2 \equiv G(\neg receive)$. Then we apply R10, where $\Psi = \{send, \neg receive\}$ and $\Psi_1 = \{send\}$. For simplicity, we abuse the notation and label the first and second R10 $\langle send\rangle$ and $\langle\neg receive\rangle$ instead of $\langle\{send\}\rangle$ and $\langle\{\neg receive\}\rangle$. The application of the other rules is straightforward.

Remark 2 In the tableau rules shown in Table 1, there is a rule labeled by $\langle\Psi_1\rangle$, but no rule is labeled by $[\Psi_1]$. The reason is that those rules are mainly dealing with existential formulae and whenever a universal formula is encountered, it is transformed to an existential one using

Table 2 Tableau of $AG(send \rightarrow F(receive))$

$\neg : AG(\neg\, send \vee F(receive))$ **(R5)**	
$\vee : EF(send \wedge G(\neg\, receive))$ **(R9)**	
$\vee : E(send, G(\neg\, receive))$ **(R9)**	$\langle true\rangle : EX(F(send \wedge G(\neg\, receive)))$ **(R10)**
$\langle send\rangle : E(send, \neg\, receive, XG(\neg\, receive))$ **(R10)**	$EF(send \wedge G(\neg\, receive))$
$\vee : EG(\neg\, receive)$ **(R9)**	
$\langle \neg\, receive\rangle : E(\neg\, receive, XG(\neg\, receive))$**(R10)**	
$EG(\neg\, receive)$	

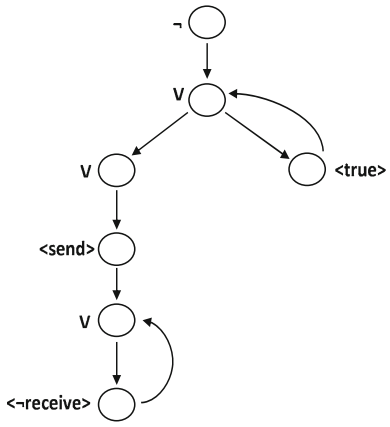

Fig. 6 arABTA of $AG(send \rightarrow F(receive))$

the rule R5. In fact, a rule labeled by $[\Psi_1]$ would be used to deal with universal formulae having the form $A(\Psi, X\varphi_1, \ldots, X\varphi_n)$ and the rule would have the form: $[\Psi_1] : \frac{A(\Psi, X\varphi_1, \ldots, X\varphi_n)}{A(\varphi_1, \ldots, \varphi_n)}$. Having the rule R5, the rule $[\Psi_1]$ would be redundant because applying this new rule followed by R5 (the only rule possible when $A(\varphi_1, \ldots, \varphi_n)$ appears) is equivalent to applying first R5 to $A(\Psi, X\varphi_1, \ldots, X\varphi_n)$ followed by $\langle\neg\Psi_1\rangle$. In both cases we will end up with $E(\neg\varphi_1, \ldots, \neg\varphi_n)$. Although the new rule is redundant, adding it will still produce arABTAs because there is no rule that can produce $A(\Psi, X\varphi_1, \ldots, X\varphi_n)$, so states labeled by $[\Psi_1]$ will not have recursive children. On the other hand, notice that it is possible to replace the rules R3, R5, R6, R7, R8, R9, and R10 by other rules dealing with universal formulae. In this case, R5 would have the form $\neg : \frac{E(\Phi)}{A(\neg\Phi)}$ and the rule $[\Psi_1]$ would be used instead of R10. Technically, this means the ABTAs obtained can contain either states labeled by $\langle\Psi_1\rangle$, or $[\Psi_1]$, but not both, which still complies with the ABTA's definition. However, using rules with universal formulae will not necessarily produce arABTAs because applying the new rule replacing R9 together with the rule $[\Psi_1]$, a node labeled by $[\Psi_1]$ will have recursive children.

Theorem 4 *The model checking problem for GCTL* can be solved in space* $O(|\psi|(|\psi| + \log|\mathcal{M}_G|)^2)$.

Proof As explained in the preliminaries, the problem of model checking GCTL* is the problem of determining if the product graph $\mathcal{B}_{\mathcal{M}_G,\psi}$ contains a successful run, which means checking the nonemptiness of the arABTA $\mathcal{B}_{\mathcal{M}_G,\psi}$. Here we present the on-the-fly algorithm presented in [10] and then we analyze its space complexity, which has not been done in [10]. The algorithm avoids the storage penalty associated with the construction of

strongly-connected components[8] and uses two depth-first searches, DFS1 and DFS2. The algorithm is a top–down marking algorithm. DFS1 recursively marks nodes as either true or false and DFS2 is lunched whenever an accepting node is found to check if the node is reachable from itself via nodes not previously traversed by DFS2. In fact, the success of DFS2 means the existence of runs with successful infinite paths. Thus, the motivation behind the requirement for nodes of not being previously traversed by DFS2 is to avoid unnecessary recomputation of successful paths already found. This is because a node N is already traversed by DFS2 if it is an accepting state and a recursive child of another accepting state which has been already found by DFS1 so that the successful infinite path to which N belongs has been already identified. When executing DFS1, some nodes are not directly marked true or false, but are marked as ***dependant*** on their recursive children, which are ***previously traversed*** by DFS1 but not marked yet, so they are marked true (resp. false) once the nodes on which they depend are marked true (resp. false). This procedure is called *mark propagation* and happens in a strongly-connected component because the nodes previously traversed can be marked later by exploring other branches in the same component. Thus, once a node N is marked true or false, the *mark is propagated* to reachable nodes from N that are marked dependant on N, which means already traversed using DFS1. This needs to record a *dependency set* for each node N. In fact, those dependant nodes (i.e., the elements of the dependency set) are the parent nodes of N that are reachable from N and already traversed. The algorithm proceeds by exploring the label of the states, which are partitioned into negative, conjunctive and disjunctive states. Negative states are those labeled by $\neg$; conjunctive states are those labeled by $\wedge$ and $[\Theta]$; and disjunctive states are those labeled by $\vee$ and $\langle\Theta\rangle$. The following recursive procedure illustrates the marking algorithm.

1. Start at the initial state.
2. At a leave (q, s, i), mark the node true if s satisfies $h(q)$; otherwise, mark the node false.
3. At a negative node, evaluate the state by recursively applying the procedure to the non recursive child, and mark the node true if the child is marked false; otherwise mark the node false.
4. At a conjunctive node N, proceed as follows:
   (a) Start by non-recursive children and evaluate the node N by applying the procedure recursively to those children. Label N false if one of the children is labeled false and propagate the mark (i.e., mark the dependant nodes on N (if any) true or false depending on the mark of the node N).
   (b) If all the children are evaluated true and there is no recursive child of the node N, then mark the node true and propagate the mark.
   (c) Otherwise, if the unique recursive child has not been already traversed, then apply the procedure recursively to this unique child and mark the node N true if the child is marked true; otherwise, mark the node false and propagate the mark.
   (d) If the recursive child has been already traversed but not market yet, then mark the node N as dependant on the recursive child.
   (e) If the node N is not marked true or false and if it is accepting, then mark N true and propagate the mark if it is reachable from itself using states not marked false. Mark N false and propagate the mark if not.
   (f) If none of the previous cases apply, mark the node false.

[8] By storage penalty, we mean the memory cost of constructing and recording the strongly-connected components of the product graph to be checked, which is needed by some automata-based model checking algorithms. As the strongly-connected components should be stored prior to any exploration by those algorithms, the memory (or space) cost is high.

5. At a disjunctive node N, proceed as follows:
   (a) Start by non-recursive children and evaluate the node N by applying the procedure recursively to those children. Label N true if one of the children is labeled true and propagate the mark.
   (b) If all the children are evaluated false and there is no recursive children of the node N, then mark the node false and propagate the mark.
   (c) Otherwise, search for a recursive child that has not been traversed yet, and if found, then apply the procedure recursively to this child and mark the node N true if the child is marked true and propagate the mark.
   (d) Otherwise, if all the recursive children are already traversed, then mark the node N as dependant on its recursive children.
   (e) If the node N is not marked true or false and if it is accepting, then mark N true and propagate the mark if it is reachable from itself using states not marked false. Mark N false and propagate the mark if not.
   (f) If none of the previous cases apply, mark the node false.

Let us now consider the complexity of this algorithm.[9] For each state (q, s, i) in the product graph $\mathcal{B}_{\mathcal{M}_G,\psi}$, if the children are already marked recursively, then marking the state becomes a problem of evaluating a Boolean expression since the children represent the subformulae of the formula in the state. As we consider Boolean expressions over the set Ver, the length of each expression is linear in the size $|\mathcal{B}_{\mathcal{M}_G,\psi}|$ of the arABTA product. As the problem of evaluating Boolean expressions is in LOGSPACE [38], marking a state assuming all the children states are marked can be done deterministically in space $O(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. Before analyzing the different cases, let us consider the propagation procedure. In fact, the property of arABTA used in this algorithm is that this propagation can be done deterministically in space $O(\log^2|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. The procedure consists in recording the dependency set, which means determining if the parent nodes of a given node N are reachable from N and already marked traversed. The reachability from N is a graph accessibility problem, and it is known by Jones [31] that the problem is in NLOGSPACE, so it can be done nondeterministically in space $O(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$, or, by Savitch's theorem [44], deterministically in space $O(\log^2|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. A necessary condition for a node to be already traversed is to be a recursive node of a given node. Thus, the size of already traversed nodes is bounded by the size of the recursive children. On the one hand, as in arABTA a node labeled by $\wedge$ has only one recursive child, and a node labeled by $[\Theta]$ has no recursive children, the size of recursive children of a conjunctive node is logarithmic in the size of the product graph $\mathcal{B}_{\mathcal{M}_G,\psi}$. On the other hand, as for a disjunctive node only one recursive child should be recorded at time, the size of recursive children needed at a given moment is also logarithmic in the size of the product graph. Thus, marking a node as already traversed can be done deterministically in space $O(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$, so the whole procedure can be done in space $O(\log^2|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. Let us now analyze the different cases. If the state is a leave, marking the state is simply evaluating a positive or negative literal, which can be done deterministically in space $O(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. If the state is a negative node, assuming the non-recursive child is evaluated, marking the node is simply complementing the evaluation of the child, so it can be done deterministically in space $O(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. Let us then consider the case of a conjunctive state (the case of a disjunctive state is symmetric). If all the children are non-recursive, then evaluating the node assuming that the children are already evaluated recursively can be done, as explained above, deterministically in space $O(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. If the node has a recursive child, which

[9] The complexity analysis of the algorithm is novel in this paper and has not been addressed in [10].

is already evaluated, then evaluating the node is simply evaluating a Boolean expression deterministically in space $O(\log |\mathcal{B}_{\mathcal{M}_G,\psi}|)$. Otherwise (i.e., the child is already traversed), the mark is propagated, and this can be done, as explained above, deterministically in space $O(\log^2 |\mathcal{B}_{\mathcal{M}_G,\psi}|)$. If the node is accepting, then marking it becomes a problem of reachability from itself using states not already marked false. Assuming the nodes are already marked, this can be done nondeterministically in space $O(\log |\mathcal{B}_{\mathcal{M}_G,\psi}|)$ [31], or, by Savitch [44], deterministically in space $O(\log^2 |\mathcal{B}_{\mathcal{M}_G,\psi}|)$.

In practice, we do not keep the Boolean values of the children, but whenever we need such a value, we evaluate it recursively. As argued in [34], the depth of the recursion is bounded by the depth of the automata, which is, from Theorem 3, $O(|\psi|)$. Thus, marking the initial state can be done deterministically in space $O(|\psi|(\log^2 |\mathcal{B}_{\mathcal{M}_G,\psi}|))$. From Proposition 4, we know that: $|\mathcal{B}_{\mathcal{M}_G,\psi}| = O(|\mathcal{M}_G| \cdot |\mathcal{B}_\psi|)$ and $|\mathcal{B}_\psi| = O(2^{|\psi|})$. Thus, the model checking problem of GCTL* can be solved in space $O(|\psi|(\log^2(|\mathcal{M}_G| \cdot 2^{|\psi|})))$, which means $O(|\psi|(|\psi| + \log |\mathcal{M}_G|)^2)$. □

Let us now discuss the explicit structure complexity of GCTL* model checking as the complexity of this problem in terms of the size of the input explicit model $\mathcal{M}_G$, that is assuming the formula fixed. In what follows, the logspace and polynomial reductions are denoted respectively by $\leq_{\log}$ and $\leq_p$.

Proposition 5 *Let $Mod(L)$ be a model of the language L, where $L \in \{CTL, CTLC^+, CTL^*, GCTL^*\}$.*

1. $Mod(CTL^*) \leq_{\log} Mod(GCTL^*)$
2. $Mod(CTL) \leq_{\log} Mod(CTLC^+)$
3. $Mod(CTLC^+) \leq_{\log} Mod(GCTL^*)$

Proof

1. CTL* is a subset of GCTL* and thus any model of CTL* is also a model of GCTL*. So, we can easily imagine a deterministic Turing machine TM that can compute this reduction in space $O(\log n)$ where n is the size of the input model of CTL*. In fact, TM simply looks at the input and writes in its output tape, one by one, the states (including the initial ones), labeling function, and transitions.
2. CTL is a subset of CTLC$^+$, so the result follows using a similar proof as 1.
3. Here we show that the model reduction presented in Sect. 3.2 can be computed by a deterministic Turing machine TM in space $O(\log n)$ where n is the size of the input model of CTLC$^+$. TM reads in the input tape a model of CTLC$^+$ and generates in the output tape, one by one, the same states including the initial ones and the same state labeling function as the input. Furthermore, TM writes α^o in the set of actions Ac if there are transitions defined in R_t, the transition relation in the model of CTLC$^+$, and reads the accessibility relations $\sim_{i\to j}$ in the input model one by one and for each one, it writes α^{ij} and β^{ij} in Ac. Then, for each element in Ac, TM writes in the output tape, l_{Ac} one by one as explained in Sect. 3.2. Finally, TM looks at each transition (s, s') in the input model and writes, one by one, the transitions (s, α^o, s'). In the same way, TM writes, one by one, the transitions (s, α^{ij}, s') and (s', β^{ij}, s) for each accessibility relation $s \sim_{i\to j} s'$ in the input model. □

Theorem 5 *The explicit structure complexity of GCTL* model checking is NLOGSPACE-complete.*

Proof

Membership: By fixing the formula ψ to be checked, we obtain an arABTA of a fixed depth. Using the algorithm presented in the proof of Theorem 4, checking the nonemptiness of this automata can be done deterministically in space $O(\log^2 |\mathcal{M}_G|)$, that is, the problem is in NLOGSPACE.
Hardness: The hardness in NLOGSPACE follows directly from Proposition 5 (i.e., $Mod(\text{CTL}^*) \leq_{\log} Mod(\text{GCTL}^*)$) as it is proven in [34] that the explicit structure complexity (called program complexity) of CTL* model checking is NLOGSPACE-complete.

□

Theorem 6 *The complexity of GCTL* model checking for concurrent programs is PSPACE-complete.*

Proof

Membership: As shown in Theorem 4, the model checking problem of GCTL* can be solved in space ***polynomial*** in the length of the formula $|\psi|$, but only ***poly-logarithmic*** in the size of the explicit model $|\mathcal{M}_G|$. Since the explicit model is obtained as the product of the components of a concurrent program and this product is at most exponentially larger than the program, the state space is exponential in the length of the program. Thus, membership in PSPACE follows from the fact that the model checking algorithm presented in the proof of Theorem 4 is on-the-fly, that is, we do not have to store all of the product automaton at once and can store, at each step, only the current configuration (a similar argument is used in [56] and [34]).
Hardness: The hardness in PSPACE is direct from the fact that CTL* $\leq_p$ GCTL* and model checking CTL* is PSPACE-complete for concurrent programs [34]. □

It is possible to prove hardness in PSPACE using a reduction from polynomial space Turing machines, for example as done in [34]. However, for the sake of simplicity, we used a direct reduction from the model checking of CTL*. We could also use a direct reduction from the nonemptiness problem of concurrent programs proven in [33] to be PSPACE-complete.

Complexity of model checking CTLC$^+$

As we did for the complexity of GCTL*, in this section, two results will be presented: (1) the explicit structure complexity of CTLC$^+$ model checking (i.e. by fixing the formula) is NLOGSPACE-complete; and (2) model checking CTLC$^+$ for concurrent programs with respect to the size of the components P_i and the length of the formula being checked is PSPACE-complete.

Theorem 7 *The explicit structure complexity of CTLC$^+$ model checking is NLOGSPACE-complete.*

Proof

Hardness: The hardness in NLOGSPACE follows directly from Proposition 5 (i.e., $Mod(\text{CTL}) \leq_{\log} Mod(\text{CTLC}^+)$) and the explicit structure complexity (called program complexity) of CTL model checking is NLOGSPACE-complete [34].
Membership: From Sect. 3.2, Theorem 2, and Proposition 5, we proved, using explicit structures, that: (1) $Mod(\text{CTLC}^+) \leq_{\log} \text{Mod}(\text{GCTL}^*)$; and (2) the reduction is sound. Thus, the membership in NLOGSPACE follows from Theorem 5. □

Theorem 8 *The complexity of CTLC$^+$ model checking for concurrent programs is PSPACE-complete.*

Proof

Hardness: The PSPACE lower bound is direct from the fact that CTL $\leq_p$ CTLC$^+$ and the complexity of model checking CTL is PSPACE-complete for concurrent programs [34].
Membership: In Sect. 3.2, we have presented a polynomial-time transformation of a model $\mathcal{M}_C$ for CTLC$^+$ to a model $\mathcal{M}_G$ for GCTL* and a formula φ_{CTLC^+} to a formula φ_{GCTL^*} so that $\mathcal{M}_C \models \varphi_{CTLC^+}$ iff $\mathcal{M}_G \models \varphi_{GCTL^*}$. Thus, since the model checking problem of GCTL* can be solved in space polynomial in the length of the formula $|\varphi_{GCTL^*}|$, and poly-logarithmic in the size of the explicit model $|\mathcal{M}_G|$ (Theorem 4), we obtain an upper bound space complexity for model checking CTLC$^+$ with regard to the length of the formula and the size of the explicit model $\mathcal{M}_C$. On the other hand, from Theorem 7, the space complexity of model checking CTLC$^+$ is poly-logarithmic in the size of the explicit model $|\mathcal{M}_C|$. And since the model checking problem of CTL can also be solved in space polynomial in the length of the formula [34], we obtain the space complexity of model checking CTLC$^+$, that is polynomial in the length of the formula $|\varphi_{CTLC^+}|$, and poly-logarithmic in the size of the explicit model $|\mathcal{M}_C|$. Thus, using a similar proof as the one presented in Theorem 6 and observing that the same on-the-fly algorithm presented in this proof can be used for CTLC$^+$ thanks to the transformation, the result follows. □

4 Case studies

One of the main motivations of this paper is to check the effectiveness of our reduction techniques and experimentally confirm the theoretical space complexity results proved so far.

We have implemented the reduction techniques presented in Sect. 3 on top of the extended NuSMV and CWB-NC. Such tools have been used to check action as well as state formulae. Specifically, NuSMV has been successfully adopted to model checking web service composition [32], multi-agent interaction protocols [21], and business models [54]. One limitation is that it does not support model checking epistemic properties in a system of agents. The extended NuSMV is used to overcome such a limitation [35,42]. It has also been used to verify commitment-based protocols [22]. CWB-NC is used for model checking large-scale protocols in agent communication [5] and security pattern composition [19]. Concretely, the reduction from CTLC$^+$ to ARCTL using the transformation function $\mathscr{F}$ is defined as a library of M4 macros. M4 is a general-purpose macro processor available on most UNIX platforms. The reduction from CTLC$^+$ to GCTL* using transformation function $\mathscr{H}$ is performed by Com-2-CWB tool we have implemented. As shown in Fig. 7, the solely manual intervention is the provision of the input file describing the problem to be verified.

On the one hand, the extended NuSMV is a symbolic model checker based on OBDDs, where the states of the model and formula to be checked are represented by means of Boolean functions, which can be easily represented using OBDDs and the set of states of the model satisfying an ARCTL formula is also represented as a Boolean function. By comparing the later set with the set of initial states represented also as a Boolean function, it is possible to establish whether or not a formula holds in a given model. As a result, the problem of model checking ARCTL is reduced to the comparison of Boolean functions. In extended NuSMV, models are described into a modular language called *extended symbolic model verifier* (extended SMV) with respect to a finite state machine formalism.

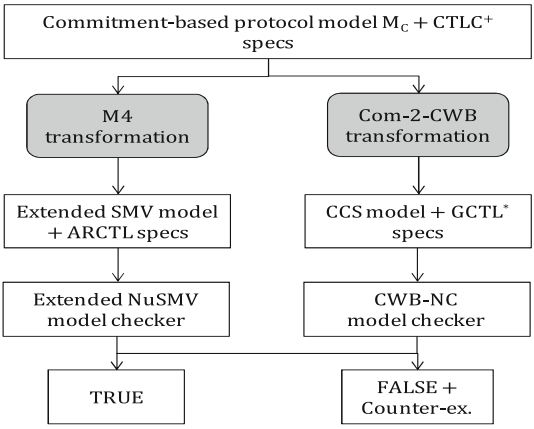

Fig. 7 Verification work flow for commitment-based protocols

On the other hand, the CWB-NC is an automata-based model checker based on ABTAs, which are a variation of alternating tree automata such as deterministic and non-deterministic Büchi automata to support efficient model checking wherein the algorithm searches only the part of the state space that needs to be explored to prove or disprove a certain formula [10]. The state space is never constructed a priori. Specifically, in CWB-NC, the model and formula are translated into ABTA and then the product graph of those ABTA are computed to check if the model is accepted by the product automaton. As a result, the problem of model checking GCTL* is reduced to check the emptiness condition of the ABTA product. In CWB-NC, the models are described into a process algebra language called calculus of communicating systems (CCS).

Two case studies for which we have been able to carry out the above motivations are the NetBill protocol (NB) [52] and Contract Net protocol (CN),[10] already applied to show how commitments can specify protocols in business settings [22,23,63].

4.1 Verifying NB protocol

The NB protocol is an electronic commerce protocol designed to be used for the selling and delivery of low-priced information goods such as software programs and journal articles over the Internet [52]. It orchestrates and regulates interactions between two agents: the merchant $Merc$ and customer $Cust$ [66,40]. In particular, the protocol starts when the customer requests a quote for some desired goods at the global state s_0 (see Fig. 8). This request is followed by the merchant's reply by sending the price quote as an offer at s_2. The customer can then either reject the offer and the protocol moves to the initial state s_0 after passing through the failure state s_4, or accept the offer, which means the customer commits to send the payment to the merchant at s_3.

Suppose that the customer agent accepts the received offer, then he has two choices: (1) to fulfill its commitment by sending the payment to the merchant at s_5, which is accessible from s_3 using $\sim_{Cust \to Merc}$; or (2) to violate its commitment and the protocol will move to s_0 after passing through the failure state s_4. When the merchant receives the payment, he commits to deliver the requested goods to the customer at s_5. In a way similar to the customer's choices, the merchant can fulfill its commitment by delivering the requested goods to the

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

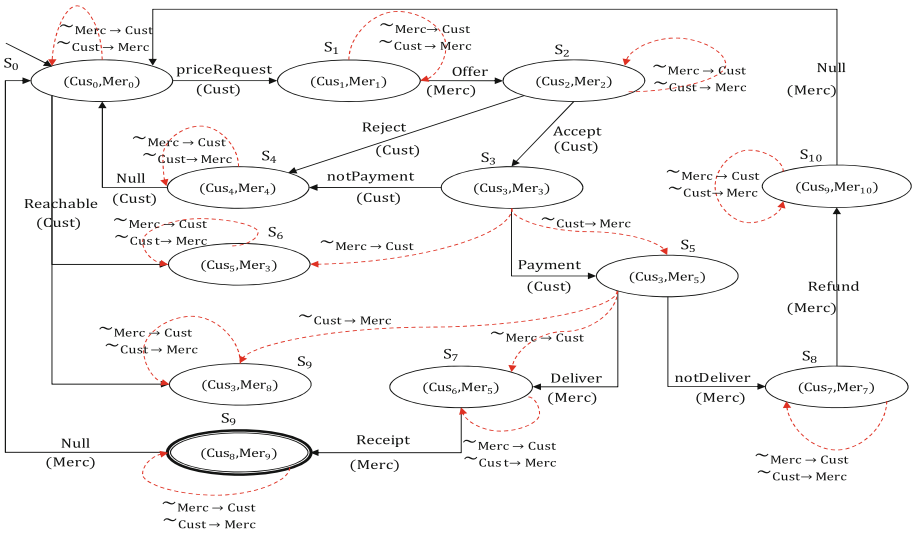

Fig. 8 The NB protocol representation using our model $\mathcal{M}_C$. Notice that each transition connecting local states is labeled by an agent and its local action. Also, the *dashed arrows* refer to the social accessibility relation using $\sim_{i\to j}$

customer at s_7, which is accessible state from s_5 using $\sim_{Merc\to Cust}$ and then moves to the acceptance state s_9 after sending the receipt to the customer and finally the protocol moves to s_0. Conversely, the customer can pay for the requested goods without being delivered by the merchant. In this case, the merchant violates its commitment at s_8 and then the protocol moves to the initial state s_0 after sending the refund to the customer at s_{10}.

This protocol can be extended to any number n of agents greater than two. We encode the NB protocol using our model $\mathcal{M}_C$=$(S, I, R_t, \{\sim_{i\to j} \mid (i, j) \in \mathcal{A}^2\}, \mathcal{V})$ by introducing n agents to represent the customers and merchants. Concretely, in the extended SMV the participating agents in the NB protocol are encoded as a set of isolated modules `MODULE anAgent<name>` in which each agent module is instantiated in the main module at run time using the `VAR` keyword, which generally defines the SMV variables. The main module also includes the `SPEC` keyword to specify the formulae that need to be checked using the ARCTL syntax. For each agent, we can use the `VAR` keyword to define its local states including the commitment, fulfillment, violation, acceptance and failure states in the form of enumeration type. The local actions of each agent are represented as input variables using the `IVAR` keyword. Agent's protocol is defined as a relation between its local state and action variables through the `TRANS` statement. The labeled transitions among states are encoded using the `TRANS` statement with the `next` and `case` expressions that represent agents's choices in a sequential manner, and initial conditions using the `INIT` statement. Figure 9 gives the main components of agent definition as an SMV module. The complete encoding of the NB protocol using the extended SMV language is available for download.[11]

Many properties can be checked in the NB protocol [5,21], such as *fairness constraints, liveness, safety, reachability, deadlock freedom, livelock freedom*. Two examples of these properties formalized using CTLC$^+$ are listed in Table 3. The first formula is given in the universal form and the second one uses the existential form. The first formula expresses an

[11] http://users.encs.concordia.ca/~bentahar/Case-Studies.zip.

```
MODULE main                                    -- main module
VAR Cust : Customer(args1,args2);              -- customer agent
    Merc : Merchant(args1,agrs2);              -- merchant agent
SPEC <formulae_list>;                          -- formulae

MODULE Customer(args1,agrs2)                   -- customer agent module
        VAR local_state: {...};                -- the local states of customer
        IVAR local_action: {...};              -- the local actions of customer
        INIT(...);                             -- initial conditions
        TRANS(local_action =
                          case ... esac);      -- the customer's protocol
         INIT(...);
         TRANS(next(local_state)=
                          case ... esac);      -- the customer's evolution function
```

Fig. 9 Example of agent structure in extended SMV. Notice that "--" defines comments in the SMV program

Table 3 Examples of tested formulae

$\varphi_1 = AG\neg\big(Fu(C_{Cust\rightarrow Merc}\, Pay) \wedge AG\neg\, C_{Merc\rightarrow Cust}\ Deliver\big)$
$\varphi_2 = EF\ Fu(C_{Merc\rightarrow Cust}\ Deliver)$

example of the standard *safety property*, i.e., "something bad never happens". In general, the safety property is expressed by AG$\neg p$ where p characterizes a "bad" situation, which should be avoided. In our protocol, a bad situation happens when the customer fulfills its commitment by sending the payment, but the merchant never commits to deliver the requested goods. The motivation behind this property is to check if the protocol is consistent, i.e., the NB protocol should not yield conflicting computations.

The formula φ_2 is an example of the standard *reachability property*, i.e., a particular situation can be reached from the initial state via some computation sequences. It states that along a given path, it is eventually the case that there is a possibility for the merchant to fulfill its commitment by delivering the requested goods. This property checks if the NB protocol is effective, i.e., the protocol's transitions should be enough to confirm that it can be executed and ended successfully.

Our experimental results were performed on a laptop equipped with the Intel(R) Core(TM) 2 Duo clocked at 1.66 GHz processor and 2 GB RAM running under Ubuntu Linux 8.04 with a vanilla 2.6.24-28-generic Kernel. We reported 5 experiments in Table 4 wherein the number of agents (# Agent), number of reachable states (# States), execution time (Time) in seconds (s), which is the summation of the time required for building all OBDDs parameters and the actual execution time for the verification, and memory in use (Memory) in MB are given. From Table 4, we notice that the number of reachable states (which reflects the state space) and execution time—especially from experiment 3—increase exponentially when the number of agents increases. In contrast, the memory usage does not increase exponentially, but only polynomially when augmenting the number of agents, which approves the complexity results presented in Sect. 3.3.

Notice that from experiment 2 we rewrite the defined formulae in a parameterized form, for example in experiment 5:

$$\varphi_1' = AG\neg\left(\bigwedge_{i=1}^{5} Fu(C_{Cust_i\rightarrow Merc}\ Pay_i)\bigwedge_{i=1}^{5} AG\neg\, C_{Merc\rightarrow Cust_i}\ Deliver_i\right)$$

Table 4 Verification results of the NB protocol using extended NuSMV

# Agents	# States	Time (s)	Memory (MB)
2	12	0.020	4.241
3	446	0.184	5.507
4	4, 224	2.736	12.957
5	33, 454	63.687	15.432
6	238, 787	630.914	83.839

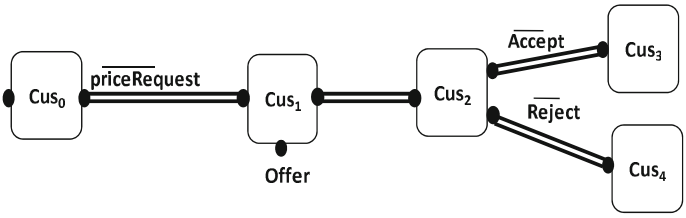

Fig. 10 Example of the interactions among processes in the *Cust* agent

This formula captures the bad situation in the NB protocol that intuitively means the merchant never commits to deliver the requested goods to none of the five customers paid for these goods.

In order to encode the NB protocol formalized using our model with the CCS language to use CWB-NC as a benchmark, we first present the syntax of CCS language using the following BNF grammar [10]:

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

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`.

A given model of the NB protocol can be encoded using the language CCS by associating each agent to a set of processes (in the sense of process algebra) wherein such processes represent the agent's local states in a recursive manner. Following standard conventions, an 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 (see Fig. 10). These channels are reliable guaranteeing timely delivery of messages. Internally, the commitment, fulfillment, violation, acceptance, and failure 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 states. For example, the customer *Cust* agent can be specified as follows:

`proc` Cus_0 `=` `'priceRequest.`Cus_1
`proc` Cus_1 `=` `Offer.`Cus_2
`proc` Cus_2 `=` `('Accept.`Cus_3 `+` `'Reject.`Cus_4`)`
`...`

Table 5 Verification results of the NB protocol using CWB-NC

# Agents	# States	Time (s)	Memory (MB)
2	325	0.035	4.020
3	6,501	1.785	10.340
4	128,327	58.872	25.470
5	N/A	N/A	N/A

which means the $Cust$ agent initially produces the price request message and evolves into the state Cus_1. The $Merc$ agent replies by sending the offer message, which makes the $Cust$ agent enter into the state Cus_2. At the state Cus_2, the $Cust$ agent is willing to produce: (1) accept message and enter into the state Cus_3; or (2) reject message and enter into the state Cus_4. The interactions among these processes are shown in Fig. 10.

Experimental results for the verification of the NB protocol are reported in Table 5. These experiments were performed on a laptop equipped with the Intel(R) Core(TM) 2 Duo clocked at 1.66 GHz processor and 2 GB RAM running under 32-bit Windows Vista. The execution time, in seconds, (i.e., the summation of the time required for building all ABTAs parameters and the actual execution time for the verification) and memory usage (in MB) are reported in Table 5. This table shows only the results of checking φ_2 because as φ_2 is satisfied and includes the existential path quantifier, it only performs on a fragment of the model, whilst φ_1, as it is satisfied and has the universal form, needs to be performed on the whole model. Notice that unlike the case with the automata-based CWB-NC, the verification results for model checking using extended NuSMV, which is based on OBDD techniques, are not affected by the structure of the formula being verified [35]. Moreover, we put "N/A" in Table 5, which means that CWB-NC is not applicable. Here again the number of reachable states and execution time increase exponentially when the number of agents increases while the memory usage increases only polynomially.

4.2 Verifying CN protocol

We consider the CN protocol designed from online business point of view to coordinate and regulate interactions among autonomous agents—as in NB protocol, we introduced n agents to represent the managers (Mgr) and participants (Prt)—which interact with each other to send results about a particular task. The protocol starts with the manager requesting proposals for a particular task. Each participant either sends a proposal message or a reject message. The manager accepts only one proposal among the received proposals and explicitly rejects the rest proposals. The participant with the accepted proposal informs the manager with the proposal results or the failure of the proposal. The protocol is self-described in Fig. 11 using our model, as we did in the previous case study. An example of safety in this protocol can be expressed by the formula φ_3 stating a bad situation where the manager fulfills its commitment by sending reply message, but the participant never commitments to deliver the results of the proposal. The formula φ_4 is an example of the standard reachability in the CN protocol, which means along a given path, there is a possibility for the participant to fulfill its commitment by sending the results performing its proposal.

$$\varphi_3 = AG\neg(Fu(C_{Mgr\rightarrow Prt}\ Reply) \wedge AG\neg\, C_{Prt\rightarrow Mgr}\ Results)$$
$$\varphi_4 = EF\ Fu(C_{Prt\rightarrow Mgr}\ Results)$$

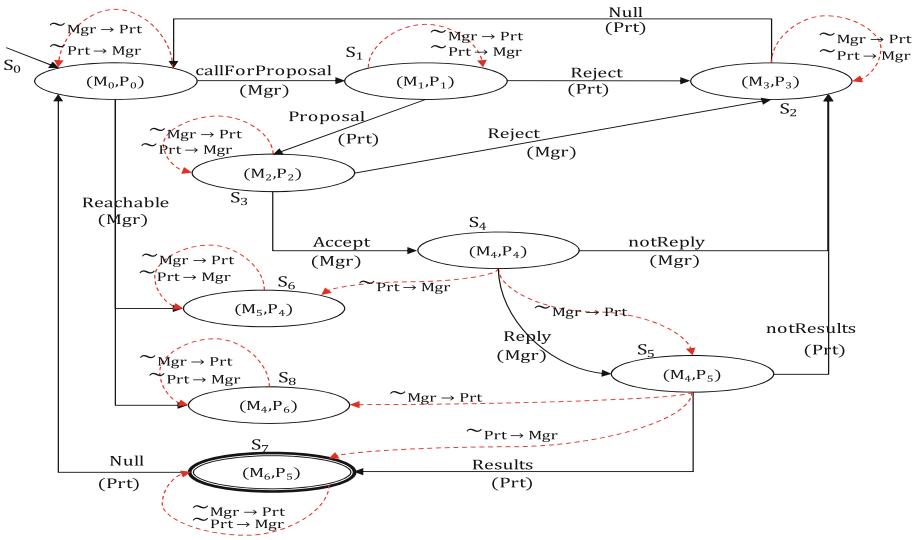

Fig. 11 Actions of the CN protocol

Table 6 Verification results of the CN protocol using extended NuSMV

# Agents	# States	Time (s)	Memory (MB)
2	9	0.044	4.226
3	528	0.1	4.734
4	4, 121	0.908	14.414
5	30, 346	11.285	14.835
6	217, 631	145.913	31.610

Table 7 Verification results of the CN protocol using CWB-NC

# Agents	# States	Time (s)	Memory (MB)
2	257	0.038	3.013
3	5,655	1.219	7.112
4	114,717	50.754	18.022
5	N/A	N/A	N/A

Table 6 shows the CN protocol verification results with extended NuSMV using the same machine as in the previous case study. Because the number of reachable states that are effectively considered in the CN protocol is much more smaller, its execution time is shorter than in the NB protocol with extended NuSMV. Table 7 reports the verification results of the CN protocol using CWB-NC. In both cases, the memory usage increases polynomially, which is expected according to the theoretical results.

We should underline that CWB-NC is efficiently applied to check satisfiability of existential formulae (i.e., for checking that a universal formula does not hold), which validate the main ABTA idea of finding counter-examples without exploring the whole model. The extended NuSMV performs moderately better than CWB-NC in terms of the execution time. It is also efficiently applicable when the model of MAS formalized using the interpreted

systems is getting larger. Furthermore, comparisons with other proposals are given in the following section.

5 Related literature

We relate our work broadly to two areas of research: (1) defining and (2) verifying social semantics of commitments and their fulfillment along with commitment-based protocols, using formal methods.

5.1 Formal semantics of commitments

Formal methods are a particular kind of mathematically-based techniques for providing rigorous frameworks to specify, model, and verify autonomous agents that communicate with one another [5,49]. They are best described as the application of formal languages using for example temporal logics, which are a set of temporal modalities allowing the specification of event orders in time without having to introduce time explicitly. Semantics generally deals with the meaning delivered by valid (syntactically correct) language constructs.

Commitments for agent communication

Singh [49] proposed a suitable formal semantics for ACLs by defining three levels of semantics for each communication act. The author extended CTL with modalities for social commitments, beliefs and intentions to represent communication among autonomous agents. He presented three accessibility relations to define the semantics of those modalities. Particularly, the accessibility relation $\mathbb{C} : \mathcal{A} \times \mathcal{A} \times \mathcal{A} \times S \rightarrow 2^{\Pi}$ where $\mathcal{A}$ is a set of agents, S is a set of states and Π is a set of paths, produces the set of accessible paths along which the commitments made at a state $s \in S$ by the debtor i towards the creditor j in the social context G hold. Thus, the semantics of the commitment modality is satisfied at s in a model $\mathcal{M}$ iff the content is true along every accessible path π defined using the accessibility relation $\mathbb{C}(i, j, G, s)$ and emanating from the commitment state s. Singh's approach also refers to a mental component by claiming that communication should be sincere.

Colombetti [16] introduced a new speech-act based ACL, which is named *Albatross* (agent language based on a treatment of social semantics). This language is based on the social notion of commitments contrary to mental states. The author proposed an *extended first order modal language* to define the semantics of Albatross. The semantics of commitments is defined using a certain type of accessibility relation: $f_c : D_{action} \times D_{agent} \times D_{agent} \times S \rightarrow 2^{2^S}$, which produces a family set of states for each state, a typed domain of individual sorts of action, and a pair of two agents. Technically, such a semantics is defined by computing the set $||\varphi||$ of states satisfying the commitment content φ and testing if this set is among the set of sets of states computed by f_c.

The first work on combing social commitments as deontic notions and arguments into the paradigm of ACL was done by Bentahar et al. [6]. In this hybrid approach, the social and public aspects of conversations can be captured by commitments and the reasoning aspects can be captured by means of arguments. In continuation of this work, Bentahar et al. [7,8] developed "Commitment and Argument Network (CAN)", which is a unified framework for pragmatic and semantics issues. It uses both a temporal logic CTL^{*CA}, which extends CTL* with modalities for commitments and their actions, argument modality, and a dynamic logic (DL) to define a logical semantics for agent communication. The authors then

introduced two accessibility relations to define the semantics of commitment and argument modalities. The accessibility relation dedicated to commitments [8] is defined as follows: $R_{sc} : \mathcal{A} \times \mathcal{A} \times S \rightarrow 2^{\Pi}$. It associates with a state s a set of accessible paths along which an agent commits towards another agent. Thus, in terms of semantics, the commitment about φ is satisfied in a model $\mathcal{M}$ at a state s iff the content φ holds along every accessible path started at s and computed by R_{sc}. Furthermore, the semantics of the *Satisfy* formula, which captures the commitment satisfaction, is defined in terms of whether a commitment has been created in the past and still active and its content holds. So, it has the same problem raised in Mallya et al.'s framework [41].

El-Menshawy et al. [21] showed how social commitments can be mapped into underlying logic-based formalisms by extending CTL* with: (1) past-directed temporal modalities; and (2) modalities for commitments and associated actions. In order to define the semantics of unconditional and conditional commitments, the authors introduced two accessibility relations. For instance, the accessibility relation devoted to unconditional commitments is defined as follows: $R_{scp} : S \times \mathcal{A} \times \mathcal{A} \rightarrow 2^{\Pi}$, which associates to a given state s a set of accessible paths along which an agent commits towards another agent. Such paths are conceived as merely possible, and as paths where the contents of commitments made in s are true. For example, if we have: $\pi \in R_{scp}(s, i, j)$, then the commitments that are made in the state s by i towards j are satisfied along the path $\pi \in \Pi^s$, where Π^s is a set of paths starting at s.

The aforementioned approaches [7,8,16,21,49] have made a significant progress in defining the formal semantics of social commitments by making use of accessibility relations. The fact that these relations are not defined and computed using the formalism of interpreted systems and agents' global and local states makes the application of our model checking-based reduction technique to these approaches hard because such a reduction technique is fundamentally based on this formalism. This means, in order to apply our technique, a transformation (or redefinition) of these accessibility relations within this formalism is needed.

Verdicchio and Colombetti [58] introduced a logical framework for the definition of ACL semantics based on the concept of social commitments by presenting a new branching time logic called CTL$^{\pm}$, which extends CTL* with "past-directed temporal operators". Then, they used CTL$^{\pm}$ to represent commitments and their fulfillment and violation. Unlike our logical model where violation is defined as a property, in [58], violation is explicitly defined. Their semantics of fulfilling commitment also differs from our semantics as they use "a truth-preserving translation" of the commitment content into a formula of "the semantics language" and past operator that points to the state at which the commitment is made.

Yolum and Singh have developed a formalism to represent and reason about commitment protocols called commitment machines using a CTL-like semantics [64]. In this formalism, commitments could be conditional and the condition is implemented using strict implication $p \rightsquigarrow q$, which requires q to hold when p holds. Although it is possible to extend our logic with the strict implication operator, its model checking using reduction has the problem that no operator in ARCTL or GCTL* we are using in our approach can support the strict implication. An appropriate approach to implement conditional commitments is the semantics proposed by Singh [50], which clearly identifies the link between the condition and content of commitments without using the strict implication. However, such semantics does not use the accessibility relation and possible-worlds semantics, so using a similar idea in CTLC^{+} is beyond the scope of this paper.

Commitments for general agent actions

Xing and Singh [62] proposed commitment patterns that accommodate revisions and exceptions to model agent interactions. The authors expressed commitment patterns as CTL formulae, but a commitment itself is defined as an abstract data type.

Mallya and Hunhs [39] and Mallya et al. [41] developed an extension of CTL with a way to describe time points and intervals to obtain a richer temporal framework to represent and reason about commitments and their actions. The semantics of the predicate *satisfied* is defined in terms of whether the DISCHARGE operation has been performed in the past or not. The authors assumed that "the DISCHARGE operation brings about p [the relevant commitment's content], and conversely, if p occurs, the DISCHARGE operation is assumed to have happened". This differs from our proposal as we defined the semantics of the fulfillment modality in terms of accessibility states from commitment state and left defining the truth conditions of the content of commitment as one of the key points of the satisfaction of the commitment modality.

To enable a rich modeling of temporal aspects of commitments, Torroni et al. [55] extended Mallya et al.'s work [41] by using "variables with domains" inside commitments. The authors showed that without such an extension, Mallya et al.'s representation of commitments does not cover some practical situations. For example, a commitment of i towards j to bring about p is going to hold at a given moment in the interval beginning at t_1 and ending at t_2 would be represented in Mallya et al.'s model as follows: $C(i, j, [t_1, t_2]p)$. This modeling enables reasoning about the temporal aspect without considering the p's meaning, but it does not specify the time at which the commitment is satisfied. In Torroni et al.'s model, p is associated with a variable, which is bound to a domain interval: $[T]p$ where $T \in [t_1, t_2]$. So, the commitment above can be written as follows: $C(i, j, [T]p), t_1 \leq T \leq t_2$. When there exists a possible value of T in the range $[t_1, t_2]$, the commitment is satisfied and this value can be used for further inferences. This commitment is violated at time t $(viol(C(i, j, [T]p, t)))$ "due to the elapsing at time t of a time interval in which p was supposed to be verified". The authors proposed a specification language called "commitment modeling language", which consists of a set of domain variables, constraints and rules. They used event calculus axioms not only for reasoning about the effects of commitment actions, but also for a static verification and compliance checking, which tracks the evolution of commitment statuses at run time by making use of reactive event calculus, which is implemented in $\mathcal{SCIFF}$, an abductive logic programming proof-procedure. As for Mallya et al.'s approach, this approach differs from ours at the semantic level (the commitment and its fulfillment are not defined as modalities). Furthermore, time in our proposal is abstract and captured using temporal operators, which allows us to use the model checking technique for the design time verification, which can complement the run time verification.

Singh [50] defined the semantics of commitment and its fulfillment by extending LTL logic with commitment modalities: "dialogical" (i.e., commitments about propositions) and "practical" (i.e., commitments about actions to be performed). This semantics is interpreted without using an accessibility relation but by using Segerberg's idea, which maps "each world into a set of set of worlds". To define the semantics of commitments, the author; 1) introduced a function that produces "a set of sets of periods for each moment and proposition"; and 2) computed the set of periods along which the content of commitment holds. So, the idea is checking if all possible ways in which the content of commitment may hold are in the set of sets of periods satisfying the condition of commitment, then the commitment holds. Thus, the author focused on identifying the link between the condition and content of commitments. As the semantics is not based on Kripke structures, verifying such a semantics using model

checking needs either defining an equivalent semantics using Kripke structures or interpreted systems; or defining a completely new model checking approach for the extended logic.

5.2 Formal verification of commitment-based protocols

Venkatraman and Singh [57] 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 operational specification of a commitment-based ACL with a minimal communicative act library was given by Fornara and Colombetti [26]. They in turn used these communicative acts having formal social semantics to define complex interaction protocols, such as the "English Auction protocol" with the help of interaction diagrams. The authors have introduced the idea of soundness condition when the concurrent combinations of communicative acts may have legal orderings, so the resulting protocol becomes verifiable without deeply giving the theoretical justification of this condition.

Limiting agents' autonomy, flexibility and reliability, as well as lacking the ability to capture real meaning of interactions are the main difficulties of traditional approaches that specify and model protocols by means of FSMs and Perti Nets. To overcome these limitations, Yolum and Singh [65] developed a declarative approach for formally specifying and flexibly executing protocols in which it is possible to capture the content of the actions through agents' commitments to one another. In fact, traditional approaches limit the flexibility of the agents in executing the protocols as modeling protocols is done in terms of action sequences, which are known a priori. The authors formalized commitment actions with the use of event calculus axioms that enable agents to reason about their actions and help protocol designers to track the evolution of commitments. Yolum and Singh [66] used an abductive event calculus planner to compute all possible paths that can be generated between an initial state and goal state regarding the protocol specification. By keeping track of agent's commitments, we can check whether the agent behaviors comply with its commitments, this technique is called static verification. Based on Yolum and Singh's approach, Chesani et al. [13] developed a run time commitment verification procedure to track the status of commitments. Our verification technique can be seen as complement to the run time verification.

Yolum [63] formalized the main generic properties that are required to help protocol designers to analyze and correct the development of commitment-based protocols by signaling possible errors and inconsistencies that arise at run time and determining the applicability of protocols. These properties are categorized into three classes: effectiveness, consistency and robustness. Yolum also developed algorithms that can be used to semi-automatically verify those properties using any available design tools. Our approach enhances Yolum's semi-automatic verification with full-automatic verification using model checking.

For MASs to be openly operative, there should be a balance between flexibility of executing protocols and verification in designing these protocols. Mallya et al. [40,41] have defined a tradeoff between flexibility and verification to be problematic. They proposed an approach for designing commitment-based protocols wherein traditional software engineering notions such as refinement and aggregation are extended to apply to protocols so that protocol designers should be able to create new protocols by refining or combining existing protocols at design time. For verification issues, they presented a "sound theory" of composing protocols using "state-similarity functions" based on the notion of "subsumption of protocols". The authors argued that the protocol that allows many computations is better

than the one that allows less computations giving more choice and flexibility in protocol execution.

The approach presented by Mallya et al. [40,41] were further complemented in three research proposals by [12], [17] and [28]. Cheng [12] and Desai et al. [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. They also presented another kind of properties called "protocol-specific properties". Their approach depends on translating the protocols into PROMELA (the input language of the SPIN automata-based model checker). Gerard and Singh [28] used the MCMAS model checker to verify the refinement of commitment-based protocols by developing a preprocessor tool that reads protocols and specifications (i.e., the refinement axioms or rules expressed in CTL) from files and then translates them into the ISPL (the input language of MCMAS).

Telang and Singh [54] used the NuSMV symbolic model checker to verify whether or not the operational model defined in the UML sequence diagrams correctly supports the business model that is aggregated from a set of business patterns. These patterns are defined in a highly abstract level based on the notion of commitments and mapped into CTL specifications. The commitment is defined as an isolated SMV module, which can be instantiated as a variable in the main module.

Compared to the above approaches, our verification is based on the formal translation of commitments and their fulfillment into ARCTL and GCTL* formulae without losing their real and concrete meanings as when representing them as simple data structures [17], processes [12] or variables [28,54]. Moreover, no experimental results and tools are presented in [17,12,28,54], but they only focused on checking the correctness of properties needed either for composition or refinement processes.

Bentahar et al. [5] presented a new verification method based on translating ACTL* formulae and protocols into ABTA to use CWB-NC where the commitment states are defined as variables and agent actions as atomic propositions using CCS language. This approach is close to our approach, but—as the mentioned approaches—it lacks a formal translation process. They also used a different encoding of the NB protocol, which only models each agent by describing its possible actions and each action is described by a set of states. However, we used the formalism of interpreted systems, which provides a general framework for modeling MASs. Technically, Bentahar et al.'s encoding allowed for the verification of 2 agents only, the experimental results shown in Table 5 report scenarios with up to 4 agents. While [5,12,17,28] and [54] show that the model checking of commitment-based protocols is feasible, in this paper we focused on fully automatic verification of commitments and their fulfillment as modal connectives as well as efficiency, applicability and complexity considerations.

6 Conclusion

The main contribution of this paper lies in presenting a new approach for reducing the problem of model checking CTLC$^+$, an extension of CTL with modalities for commitments and their fulfillment, into the problem of model checking ARCTL and GCTL*. In this approach, the commitment and fulfillment modalities are transformed into ARCTL and GCTL* formulae. We also computed the space complexity of model checking CTLC$^+$ with regard to explicit models and concurrent programs, which is respectively NLOGSPACE-complete and

PSPACE-complete. Furthermore, we proved the soundness of the proposed reduction techniques. Using two business protocols, we have experimentally evaluated the effectiveness and efficiency of our reduction techniques and our verification approach implemented using two different model checkers (extended NuSMV and CWB-NC). These experiments paint the following picture: the model checkers were able to verify a variety of complex formulae correctly and efficiently. Our approach is clearly not exhaustive but helps protocol designers check the compliance of protocols against given properties expressed in our logic. This paper establishes the practical usability of the approach by applying it to a large business protocol having approximately 2.3e+06 states thanks to the OBDDs-based symbolic encodings used in extended NuSMV. The overall conclusion coincides with the usual considerations in that automatic verification methods complement other static verification methods very well. When comparing our approach to other available proposals in the literature, we found that this approach considerably simplifies the specifications to be checked and maintains the feasibility of different model checking techniques.

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 consider other commitment actions, such as withdraw, assign, and delegate. Analyzing the relations between agent communication commitments and commitments in strategic logics such as the ones studied in [1] is another direction for future work. Finally, Jones and Parent [30] proposed a new approach to ACLs, which is neither intention-based nor commitment-based, but convention-based. The idea is to propose an alternative social semantics for ACLs based on conventions and conventional signals. The approach has been applied to analyze sequences of exchanges within communication protocols. As future work, we aim to investigate this alternative approach to agent communication from the model checking perspective using reduction techniques similar to the ones presented in this paper.

Acknowledgments We would like to thank the three anonymous reviewers for their valuable and very professional comments and suggestions for improvements. We also would like to thank Professor Orna Kupferman for her explanations and suggestions on the complexity part. The first and third authors thank the Ministry of Higher Education, Egypt for its financial support. The second and fourth authors are supported by NSERC (Canada). The second author is also supported by FQRSC and FQRNT (Québec).

References

1. Ågotnes, T., Goranko, V., & Jamroga, W. (2008). *Strategic commitment and release in logics for multi-agent systems (Extended abstract)*. Technical Report IfI-08-01, Clausthal University of Technology, Clausthal-Zellerfeld.
2. Artikis, A., & Pitt, J. V. (2009). Specifying open agent systems: A survey. In A. Artikis, G. Picard, & L. Vercouter (Eds.), *ESAW, LNCS* (Vol. 5485, pp. 29–45). Heidelberg: Springer.
3. Baldoni, M., Baroglio, C., & Marengo, E. (2010). Behavior oriented commitment-based protocols. In H. Coelho, R. Studer, & M. Wooldridge (Eds.), *ECAI* (Vol. 215, pp. 137–142). Amsterdam: IOS Press.
4. Bentahar, J., Meyer, J.-J. Ch., & Wan, W. (2009). Model checking communicative agent-based systems. *Knowledge-Based Systems, 22*(3), 142–159.
5. Bentahar, J., Meyer, J.-J. Ch., & Wan, W. (2010). Model checking agent communication. In M. Dastani, K. V. Hindriks, & J.-J. Ch. Meyer (Eds.), *Specification and verification of multi-agent systems* (1st edn., Chap. 3, pp. 67–102). Heidelberg: Springer.
6. Bentahar, J., Moulin, B., & Chaib-draa, B. (2004). Commitment and argument network: A new formalism for agent communication. In F. Dignum (Ed.), *ACL 2003, LNCS* (Vol. 2922, pp. 146–165). Heidelberg: Springer.

7. Bentahar, J., Moulin, B., Meyer, J.-J. Ch., & Chaib-draa, B. (2004). A logical model for commitment and argument network for agent communication. In *Proceedings of the 3rd International Joint Conference on AAMAS* (pp. 792–799). Washington, DC: IEEE Computer Society.
8. Bentahar, J., Moulin, B., Meyer, J.-J. Ch., & Lespérance, Y. (2007). A new logical semantics for agent communication. In K. Inoue, K. Satoh, & F. Toni (Eds.), *CLIMA VII, LNCS* (Vol. 4371, pp. 151–170). Heidelberg: Springer.
9. Bhat, G. (1998). *Tableau-based approaches to model-checking*. Ph.D. thesis, Department of Computer Science, North Carolina State University, Raleigh, NC.
10. Bhat, G., Cleavel, R., & Groce, A. (2001). Efficient model checking via Büchi tableau automata. In G. Berry, H. Comon, & A. Finkel (Eds.), *CAV, LNCS* (Vol. 2102, pp. 38–52). Heidelberg: Springer.
11. Castelfranchi, C. (1995). Commitments: From individual intentions to groups and organizations. In V. R. Lesser & L. Gasser (Eds.), *ICMAS* (pp. 41–48). Cambridge, MA: MIT Press.
12. Cheng, Z. (2006). *Verifying commitment-based business protocols and their compositions: Model checking using promela and spin*. Ph.D. thesis, North Carolina State University, Raleigh, NC.
13. Chesani, F., Mello, P., Montali, M., & Torroni, P. (2009). Commitment tracking via the reactive event calculus. In C. Boutilier (Ed.), *IJCAI*, Pasadena, CA, pp. 91–96.
14. Chopra, A. K., & Singh, M. P. (2009). Multiagent commitment alignment. In C. Sierra, C. Castelfranchi, K. S. Decker, & J. S. Sichman (Eds.), *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Budapest, pp. 937–944.
15. Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model checking*. Cambridge, MA: MIT Press.
16. Colombetti, M. (2000). A commitment-based approach to agent speech acts and conversations. In *Proceedings of the Fourth International Conference on Autonomous Agents, Workshop on Agent Languages and Conversation Policies*, Barcelona, pp. 21–29.
17. Desai, N., Cheng, Z., Chopra, A. K., & Singh, M. (2007). Toward verification of commitment protocols and their compositions. In E. H. Durfee, M. Yokoo, M. N. Huhns, & O. Shehory (Eds.), *AAMAS* (pp. 144–146). Richland, SC: IFAAMAS.
18. Desai, N., Chopra, A. K., & Singh, M. P. (2009). Amoeba: A methodology for modeling and evolution of cross-organizational business processes. *ACM Transaction on Software Engineering and Methodology, 19*(2), 1–40.
19. Dong, J., Peng, T., & Zhao, Y. (2010). Automated verification of security pattern compositions. *Information & Software Technology, 52*(3), 274–295.
20. El-Menshawy, M., Bentahar, J., & Dssouli, R. (2010). Modeling and verifying business interactions via commitments and dialogue actions. In P. Jedrzejowicz, N. T. Nguyen, R. J. Howlett, & L. C. Jain (Eds.), *KES-AMSTA (2), LNCS* (Vol. 6071, pp. 11–21). Heidelberg: Springer.
21. El-Menshawy, M., Bentahar, J., & Dssouli, R. (2010). Verifiable semantic model for agent interactions using social commitments. In M. Dastani, A. E. Fallah-Seghrouchni, J. Leite, & P. Torroni (Eds.), *LADS, LNCS* (Vol. 6039, pp. 128–152). Heidelberg: Springer.
22. El-Menshawy, M., Bentahar, J., & Dssouli, R. (2011). Symbolic model checking commitment protocols using reduction. In A. Omicini, S. Sardina, & W. Vasconcelos (Eds.), *DALT, LNAI* (Vol. 6619, pp. 185–203). Heidelberg: Springer.
23. El-Menshawy, M., Benthar, J., Qu, H., & Dssouli, R. (2011). On the verification of social commitments and time. In *Proceedings of the 10th International Conference on AAMAS*, Taipei, pp. 483–890.
24. Emerson, E. A., Mok, A. K., Sistla, A. P., & Srinivasan, J. (1992). Quantitative temporal reasoning. *Journal of Real-Time Systems, 4*(4), 331–352.
25. Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. Y. (1995). *Reasoning about knowledge*. Cambridge, MA: MIT Press.
26. Fornara, N., & Colombetti, M. (2002). Operational specification of a commitment-based agent communication language. In *Proceedings of the 1st International Conference on AAMAS* (pp. 535–542). New York: ACM.
27. Fornara, N., Viganò, F., Verdicchio, M., & Colombetti, M. (2008). Artificial institutions: A model of institutional reality for open multi-agent systems. *Artificial Intelligence and Law, 16*(1), 89–105.
28. Gerard, S. N., & Singh, M. P. (2011). Formalizing and verifying protocol refinements. *ACM Transactions on Intelligent Systems and Technology, 2*(3) (in press).
29. Jamroga, W., & Ågotnes, T. (2007). Modular interpreted systems. In *Proceedings of the 6th International Conference on AAMAS* (pp. 131:1–131:8). New York: ACM.
30. Jones, A. J. I., & Parent, X. (2007). A convention-based approach to agent communication languages. *Group Decision and Negotiation, 16*(2), 101–141.
31. Jones, N. D. (1975). Space-bounded reducibility among combinatorial problems. *Computer and System Sciences, 11*(1), 68–85.

32. Kova, M., Bentahar, J., Maamar, Z., & Yahyaoui, H. (2009). A formal verification approach of conversations in composite web services using NuSMV. In H. Fujita & V. Marík (Eds.), *SoMeT* (Vol. 199, pp. 245–261). Amsterdam: IOS Press.
33. Kozen, D. (1977). Lower bounds for natural proof systems. In *Proceedings of the 18th IEEE Symposium on Foundation of Computer Science*, Providence, RI, pp. 254–266.
34. Kupferman, O., Vardi, M., & Wolper, P. (2000). An automata-theoretic approach to branching-time model checking. *Journal of the ACM, 47*(2), 312–360.
35. Lomuscio, A., Pecheur, C., & Raimondi, F. (2007) Automatic verification of knowledge and time with NuSMV. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (pp. 1384–1389). San Francisco: Morgan Kaufmann.
36. Lomuscio, A., Penczek, W., & Qu, H. (2010). Partial order reductions for model checking temporal-epistemic logics over interleaved multi-agent systems. *Fundamenta Informaticae, 101*(1–2), 71–90.
37. Lomuscio, A., Qu, H., & Raimondi, F. (2009). MCMAS: A model checker for the verification of multi-agent systems. In A. Bouajjani & O. Maler (Eds.), *CAV, LNCS* (Vol. 5643, pp. 682–688). : Springer.
38. Lynch, N. (1977). Log space recognition and translation of parenthesis languages. *Journal of ACM, 24*(4), 583–590.
39. Mallya, A. U., & Huhns, M. N. (2003). Commitments among agents. *IEEE Internet Computing, 7*(4), 90–93.
40. Mallya, A. U., & Singh, M. P. (2007). An algebra for commitment protocols. *Autonomous Agents and Multi-Agent Systems, 14*(2), 143–163.
41. Mallya, A. U., Yolum, P., & Singh, M. P. (2004). Resolving commitments among autonomous agents. In Dignum F. (Ed.), *ACL 2003, LNCS* (Vol. 2922, 166–182). Heidelberg: Springer.
42. Pecheur, C., & Raimondi, F. (2007). Symbolic model checking of logics with actions. In S. Edelkamp, & A. Lomuscio (Eds.), *Model checking and artificial intelligence (MoChArt 2006), LNCS* (Vol. 4428, pp. 113–128). Heidelberg: Springer.
43. Penczek, W., & Lomuscio, A. (2003). Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae, 55*(2), 167–185.
44. Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Computer and System Sciences, 4*(2), 177–192.
45. Searle, J. R. (1969). *Speech acts: An essay in the philosophy of language*. Cambridge: Cambridge University Press.
46. Singh, M. P. (1996). *A conceptual analysis of commitments in multi-agent systems*. Technical Report, North Carolina State University, Raleigh ,NC.
47. Singh, M. P. (1998). Agent communication languages: Rethinking the principles. *IEEE Computer, 31*(12), 40–47.
48. Singh, M. P. (1999). An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law, 7*(1), 97–113.
49. Singh, M. P. (2000). A social semantics for agent communication languages. In F. Dignum & M. Greaves (Eds.), *Issues in agent communication, LNCS* (Vol. 1919, pp. 31–45). Heidelberg: Springer.
50. Singh, M. P. (2008). Semantical considerations on dialectical and practical commitments. In D. Fox & C. P. Gomes (Eds.), *AAAI* (pp. 176–181). Menlo Park, CA: AAAI Press.
51. Singh, M. P., Chopra, A. K., & Desai, N. (2009). Commitment-based service-oriented architecture. *IEEE Computer, 42*(11), 72–79.
52. Sirbu, M. A. (1997). Credits and debits on the Internet. *IEEE Spectrum, 34*(2), 23–29.
53. Telang, P., & Singh, M. (2009). Enhancing tropos with commitments: A business meta-model and methodology. In A. Borgida, V. K. Chaudhri, P. Giorgini, & E. S. K. Yu (Eds.), *Conceptual modeling: Foundations and applications, LNCS* (Vol. 5600, pp. 417–435). Heidelberg: Springer.
54. Telang, P. R., & Singh, M. P. (2011). Specifying and verifying cross-organizational business models: An agent-oriented approach. *IEEE Transactions on Services Computing, 4* (in press).
55. Torroni, P., Chesani, F., Mello, P., & Montali, M. (2010). Social commitments in time: Satisfied or compensated. In M. Baldoni, J. Bentahar, van M. B. Riemsdijk, & J. Lloyd (Eds.), *DALT, LNCS* (Vol. 5948, pp. 228–243). Heidelberg: Springer.
56. Vardi, M. Y., & Wolper, P. (1994). Reasoning about infinite computations. *Information and Computation, 115*(1), 1–37.
57. Venkatraman, M., & Singh, M. P. (1999). Verifying compliance with commitment protocols: Enabling open web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems, 2*(3), 217–236.
58. Verdicchio, M., & Colombetti, M. (2003). A logical model of social commitment for agent communication. In *Proceedings of the 2nd International Conference on AAMAS* (pp. 528–535). New York: ACM.

59. Winikoff, M. (2007). Implementing commitment-based interactions. In E. Durfee, M. Yokoo, M. Huhns, & O. Shehory (Eds.), AAMAS, Honolulu, HI, pp. 873–880.
60. Wooldridge, M. (2002). *An introduction to multi-agent system*. New York: Wiley.
61. Wooldridge, M. (2009). *An introduction to multiagent systems*. New York: Wileys.
62. Xing, J., & Singh, M. P. (2003). Engineering commitment-based multi-agent systems: A temporal logic approach. In *Proceedings of the 2nd International Conference on AAMAS* (pp. 891–898). New York: ACM.
63. Yolum, P. (2007). Design time analysis of multi-agent protocols. *Data and Knowledge Engineering, 63*, 137–154.
64. Yolum, P., & Singh, M. P. (2002). Commitment machines. In J.-J. Ch. Meyer & M. Tambe (Eds.), *ATAL, LNCS* (Vol. 2333, pp. 235–247). Heidelberg: Springer.
65. Yolum, P., & Singh, M. P. (2002). Flexible protocol specification and execution: Applying event calculus planning using commitment. In *Proceedings of the 1st International Conference on AAMAS* (pp. 527–534). New York: ACM.
66. Yolum, P., & Singh, M. P. (2004). Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence, 42*(1–3), 227–253.