Knowledge-Based Systems 22 (2009) 142–159

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Model checking communicative agent-based systems

Jamal Bentahar [a,*], John-Jules Meyer [b], Wei Wan [a]

[a] Concordia University, Concordia Institute for Information Systems Engineering, Montreal, QC, Canada
[b] Utrecht University, Department of Information and Computer Science, The Netherlands

ARTICLE INFO

Article history:
Available online 7 December 2008

Keywords:
Multi-agent systems
Model checking
Temporal logic
Agent-based negotiation protocol

ABSTRACT

Model checking is a formal technique used to verify communication protocols against given properties. In this paper, we propose a new model checking algorithm aims at verifying systems designed as a set of autonomous interacting agents. These software agents are equipped with knowledge and beliefs and interact with each other according to protocols governed by a set of logical rules. We present a tableau-ased version of this algorithm and provide the soundness, completeness, termination and complexity results. A case study about an agent-based negotiation protocol and its implementation are also described.

1. Introduction

Although formal verification methods are not yet widely used to check large and complex systems, they are useful for the verification of some properties in relatively small systems involving concurrency and communication protocols. *Deadlock* (two or more processes are each waiting for another to release a resource), *safety* (same bad situation may never occur), and *reachability* (some particular situation can be reached) are examples of such properties. Formal methods offer a potential to obtain an early integration of verification in the design process, and to reduce the verification time [17]. However, they are only applicable for finite state systems and they generally operate on system models and not on the actual system.

We distinguish two main formal verification approaches: proof-based approaches and model-based approaches. In the proof-based approaches, the system description is a set of logical formulae Γ and the specification is another formula ϕ. The verification method consists of trying to find a proof that $\Gamma \vdash \phi$. This typically requires guidance and expertise from the user in order to identify suitable lemmas and auxiliary assertions. In the model-based approaches, also called *model checking*, the system (the protocol) is represented by a finite model M modeled as a *Kripke structure* using an appropriate logic. The specification is again represented by a formula ϕ expressed in the same logic, and the verification method consists of computing whether the model M satisfies ϕ or not ($M \models \phi$). This is an algorithmic-based technique and usually done automatically.

Recently, model checking has been used to verify agent-based systems [10,11,18,21]. Verifying these systems is becoming more and more necessary because they are increasingly used in several critical application domains, such as e-commerce, simulation, distributed collaborative systems, etc [16,20,24]. In these systems, agents are equipped with reasoning and communicative abilities. This is generally expressed using epistemic logics (logics about beliefs, knowledge, goals, etc.) and logic-based protocols [2]. Such protocols are modeled as a set of rules describing the allowed communicative acts in different situations. How an agent selects the communicative act to perform at a given moment is decided by his reasoning. In order to allow agents to flexibly and autonomously communicate within multi-agent systems, these protocols are specified as a set of policies, called dialogue games, about which agents can reason [3,6,22,25]. Dialogue games are interaction games in which each agent plays a move in turn by performing utterances according to a predefined set of logical rules. Dialogue game protocols are a combination of different dialogue games.

1.1. Contributions

In this paper, we present an efficient *model checking algorithm* to verify interacting agent-based systems, in which agents have knowledge and beliefs and communicate by combining and reasoning about dialogue games. This algorithm is based on a tableau-based technique we developed in [5]. In this paper, we focus on the algorithmic description of this technique and on its termination and complexity. We specify a dialogue game protocol

* Corresponding author. Tel.: +1 514 848 2424; fax: +1 514 848 3171.
E-mail address: bentahar@ciise.concordia.ca (J. Bentahar).

0950-7051/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2008.11.006

as a transition system in which transitions are labeled with communicative acts. Such acts are modeled as actions performed by agents *on propositional commitments* (*PC*), for example, creating, accepting, or challenging a propositional commitment [6]. Propositional commitments are used to capture the *public utterances* in the sense that each utterance is viewed as a propositional commitment.

To use model checking, we specify dialogue game protocols and the properties to be verified in a new logic extending CTL* by adding formulae representing propositional commitments. The verification method is based on the translation of formulae into a variant of alternating tree automata called *alternating Büchi tableau automata* (ABTA) [9]. Unlike the model checking algorithms proposed in the literature, our *on-the-fly efficient algorithm* uses only one depth-first search instead of two. This is due to the fact that our algorithm explores directly the product graph of the dialogue game protocol and the ABTA representing the property to be verified using the sign of the nodes.

To summarize, this paper presents four contributions:

1. A new temporal logic with syntax and semantics for agent communication augmented with argumentation and actions. This logic has four new operators: argument operator, commitment operator, and two operators for creating and acting on commitments. This logic allows agents to reason about their communicative actions and justify their choices using arguments.
2. An on-the-fly model checking technique for dialogue game-based agent communication. The algorithm is on-the-fly because it does not need to build and explore the whole model like in classical automata-based model checking, but develops parts of this model as needed to check the formula. Consequently, our algorithm does not suffer so much from the state explosion problem and can be used to check scalable cases. In addition, the algorithm is proven to be linear with the size of the system (Section 7), which provides another solution to the explosion problem since partial verification is allowed with model checking (formulas can be checked partially).
3. Termination, soundness and completeness proofs for the proposed technique. Furthermore, the complexity results show the efficiency and feasibility of the approach.
4. The proposed technique can be used to check not only the soundness of agent communication protocols in terms of properties satisfaction (such as *deadlock*, *liveness*, *safety*, and *reachability*), but also the compliance of heterogeneous agents with the underlying syntax and structure of dialogue game protocols. The second verification is of great importance for a form of interoperability of multi-agent systems. Furthermore, two types of formulas are considered: temporal and action formulas.

1.2. Paper overview

The rest of the paper is organized as follows. In Section 2, we summarize and discuss related work. In Section 3, we develop a logic for communicating agents that we use to specify the properties to be checked. Section 4 presents the tableau rules associated to this logic. Section 5 presents the specification of dialogue game protocols agents use to communicate and gives some examples of the properties to be checked. In Section 6, we discuss the model checking technique of these protocols using tableau rules. In Section 7, we prove the termination property of the technique and we discuss its complexity. Before concluding the paper in Section 9, we present a case study in Section 8 describing the verification of an agent-based negotiation protocol and some issues of its implementation.

2. Related work

Bernholtz et al. [7] argued that *alternating tree automata* are the key to a comprehensive and satisfactory automata-theoretic framework for branching temporal logics. Alternating tree automata on infinite trees generalize the standard notion of non-deterministic tree automata by allowing several successor states to go down along the same branch of the tree. Tree automata generalize sequential automata in the following way: on a given binary tree, the automaton starts its computation at the root in an initial state and then simultaneously works down the paths of the tree level by level. The transition relation specifies the two states that are the two sons of a node. The tree automaton accepts the tree if there is a run built up in this fashion which is *successful*. A run is successful if all its paths are successful in a sense given by an acceptance condition for sequential automata.

The model checking approach we use in this paper is based on an alternative view of model checking proposed by Bhat and Cleaveland [8] and Bhat et al. [9]. This view relies on translating formulae into intermediate structures, ABTA. Unlike the other model checking techniques, this technique allows us to verify not only temporal formulae, but also action formulae. Because our logic for communicating agents is based on an action theory, this technique is more suitable. This approach is called *tableau-based model checking* [5].

Recently, the verification of agent-based systems has become an attractive field of research and several proposals have been put forward. Some of these proposals use existing model checkers (for example SPIN and JPF2) by translating some agent specification languages (for example MABLE and AgentSpeak) to the languages used by these model checkers [10,11,19]. Other proposals adapt some model checking techniques (for example bounded and unbounded model checking) and propose new algorithms for verifying temporal and epistemic properties [20,21,24]. Giordano and her colleagues [18] addressed the problem of specifying and verifying systems of communicating agents in a dynamic linear time temporal logic (DLTL). Our approach is fundamentally different from the approaches based on existing model checkers such as SPIN. In the particular case of SPIN model checker, the communication protocol should be designed in propositional linear temporal logic (PLTL), which is the logic supported by this model checker. Expressiveness of this language is much more less than our logic's expressiveness. Particularly, PLTL considers only linear properties and cannot be used for non-deterministic choices. Because our logic is branching-time-like, non-deterministic properties can be designed. Also, PLTL formulas are path formulas and state formulas (formulas checked on states) are not easily manageable, while in our logic, both path and state formulas are included. Furthermore, unlike our algorithm, SPIN cannot be used to check properties where argument and action operators are used. Another important difference between our approach and SPIN is efficiency as our technique uses an efficient on-the-fly algorithm.

Except the work done in [18], all the other proposals on model checking of agent-based systems are based only on temporal and epistemic logics. In this paper, we propose a model checking-based verification of dialogue game protocols using a temporal and dynamic logic. In contrast to [18], the dynamic aspect of our logic is represented by action formulae and not by strengthening the *until* operator by indexing it with the regular programs of dynamic logic. Our protocols are specified as actions that agents apply to propositional commitments. In addition, the model checking procedure that we propose allows us to verify not only that the dialogue game protocol (the theoretical model) satisfies a given property, but also that the tableau semantics of the communicative acts is respected. The idea is to integrate this semantics in the specification of the protocol, and then to propose a parsing method to verify that the protocol specification respects the semantic definition. Consequently, if agents respect these protocols, then they also respect the semantics of the communicative acts. We have here a mechanism for checking the agents' compliance with the semantics without taking into account the agents' specifications created by the developers. Indeed, we have only one procedure to verify: (1) the correctness of the protocols relative to the properties that the protocols should satisfy and (2) the conformance of agents to the semantics of the communicative acts. The purpose of this technique is to verify the temporal properties of the protocol and to ensure that the structures of the communicative acts are the same in both the protocol and the specification. Consequently, the protocol correctness is checked by verifying that the protocol satisfies the required properties, such as deadlock freedom, safety, and reachability, and by checking that unexpected sequences, expressed as logical formulas, can never happen when executing the protocol.

To our knowledge, until now there is no work that addressed the soundness verification issue of communicating agents using dialogue game protocols in the sense that these protocols satisfy the required properties. The contribution of this paper is an efficient algorithm for model checking dialogue game protocols for software interacting agent-based systems. The algorithm's termination and soundness proofs are also provided. We notice that this verification issue is different from the interoperability problem which focuses on the reliability communication between agents in the sense that agents are acting according to a given semantics [2]. Although this interoperability issue is of a capital importance for agent communication, it is not considered in this paper because our objective is to focus on the protocols' soundness.

3. A logic for communicating agents

3.1. Syntax

In this section, we present CTL^{*CA} an extended logic from CTL* for communicative agents. This logic extends CTL* by adding propositional commitments and action formulae. In what follows we use $p, p_1, p_2, \ldots$ to range over the set of atomic propositions Φ_p. The syntax of this logic is as follows:

$$\begin{aligned}
\mathscr{S} &::= p \mid \neg\mathscr{S} \mid \mathscr{S} \wedge \mathscr{S} \mid \mathscr{S} \vee \mathscr{S} \mid A\mathscr{P} \mid E\mathscr{P}\\
\mathscr{P} &::= \mathscr{S} \mid \mathscr{P} \wedge \mathscr{P} \mid \mathscr{P} \vee \mathscr{P} \mid X^+\mathscr{P} \mid X^-\mathscr{P} \mid \mathscr{P}U^+\mathscr{P} \mid \mathscr{P}U^-\mathscr{P} \mid \mathscr{P} \therefore \mathscr{P}\\
&\mid PC(Ag_1, Ag_2, t, \mathscr{P})\\
&\mid C(Ag_1, PC(Ag_1, Ag_2, t, \mathscr{P}))\\
&\mid Act(Ag_i, PC(Ag_1, Ag_2, t, \mathscr{P}))
\end{aligned}$$

The formulae generated by $\mathscr{S}$ are called state formulae, while those generated by $\mathscr{P}$ are called path formulae. We use $\psi, \psi_1, \psi_2, \ldots$ to range over state formulae and $\phi, \phi_1, \phi_2, \ldots$ to range over path formulae. The meaning of most of the constructs is straightforward (from CTL* with next (X^+), previous (X^-), until (U^+), and since (U^-) operators). The formula $\phi_1 \phi_2$ means that ϕ_1 is an argument for ϕ_2. We can read this formula: ϕ_1, so ϕ_2. This operator introduces argumentation as a logical relation between path formulae.

The formula $PC(Ag_1, Ag_2, t, \mathscr{P})$ is the propositional commitment made by agent Ag_1 at the moment t towards agent Ag_2 that the path formula $\mathscr{P}$ is true. The formula $C(Ag_1, PC(Ag_1, Ag_2, t, \mathscr{P}))$ means that agent Ag_1 creates the commitment $PC(Ag_1, Ag_2, t, \mathscr{P}) \cdot Act(Ag_i, PC(Ag_1, Ag_2, t, \mathscr{P}))$ means that agent Ag_i ($i \in \{1,2\}$) performs an action on the propositional commitment made by Ag_1 towards Ag_2. The set of actions performed on propositional commitments are *Withdraw*, *Satisfy*, *Violate*, *Reactivate*, *Challenge*, *Accept*, *Refuse*, *Justify*, *Attack*, *Defend* (see [3] for more details).

3.2. Semantics

The formal model M associated to this logic corresponds to the dialogue game protocol agents use to communicate. Formally, this model is defined as follows: $M = \langle S_m, Lab_m, Act_m, \overset{Act_m}{\rightarrow}, Agt, R_{PC}, S_{m_0} \rangle$ where: S_m is a set of states; $Lab_m : S_m \rightarrow 2^{\Phi_p}$ is the labeling state function; Act_m is the set of actions performed on propositional commitments; $\overset{Act_m}{\rightarrow} \subseteq S_m \times Act_m \times S_m$ is the transition relation; Agt is a set of agents; R_{PC}:$S_m \times Agt \times Agt \rightarrow 2^{\sigma}$ with σ is the set of all paths in M, is an accessibility modal relation that associates to a state s_m the set of paths representing the propositional commitment along which an agent can commit towards another agent; s_{m_0} is the start state. The paths that path formulae are interpreted over have the form $x^i = s_{m_i} \overset{\alpha_{i+1}}{\rightarrow} s_{m_{i+1}} \overset{\alpha_{i+2}}{\rightarrow} s_{m_{i+2}} \ldots$ where $x^i \in \sigma$, $s_{m_i}, s_{m_{i+1}}, \ldots$ are states and $\alpha_{i+1}, \alpha_{i+2}, \ldots$ are actions.

The semantics of CTL^{*CA} state formulae is as usual (semantics of CTL*). A path satisfies a state formula if the initial state in the path does. Along a path x^i, $\phi_1\phi_2$ holds if ϕ_1 is true and at next time if ϕ_1 is true then ϕ_2 is true. Formally:

$$x^i \models_M \phi_1 \therefore \phi_2 \text{ iff } x^i \models_M \phi_1 \text{ and } x^{i+1} \models_M \phi_1 \Rightarrow \phi_2$$

A path x^i satisfies $PC(Ag_1, Ag_2, t, \phi)$ if every accessible path to Ag_1 towards Ag_2 from the first state of the path using R_{PC} satisfies ϕ. Formally:

$$\begin{aligned}
&x^i \models_M PC(Ag_1, Ag_2, t, \phi)) \text{ iff}\\
&\forall x^j \in \sigma, x^j \in R_{PC}(s_{m_i}, Ag_1, Ag_2) \Rightarrow x^j \models_M \phi
\end{aligned}$$

A path x^i satisfies $C(Ag_1, PC(Ag_1, Ag_2, t, \phi))$ if C is in the label of the first transition on this path and $PC(Ag_1, Ag_2, t, \phi)$ holds along the path x^{i+1}. Formally:

$$\begin{aligned}
&x^i \models_M C(Ag_1, PC(Ag_1, Ag_2, t, \phi)) \text{ iff}\\
&\alpha_{i+1} = C \& x^{i+1} \models_M PC(Ag_1, Ag_2, t, \phi)
\end{aligned}$$

A path x^i satisfies $Act(Ag_i, PC(Ag_1, Ag_2, t, \phi))$ if Act is in the label of the first transition on this path and if in the past (P)Ag_1 has already created the social commitment. Formally:

$$\begin{aligned}
&x^i \models_M Act(Ag_1, PC(Ag_1, Ag_2, t, \phi)) iff\, \alpha_{i+1}\\
&= Act \text{ and } P(C(Ag_1, PC(Ag_1, Ag_2, t, \phi)))
\end{aligned}$$

We notice that the past (P) and future (F) operators are abbreviations from until operator (U) in the usual way of CTL* logic.

4. Tableau rules for CTL^{*CA}

Tableau-based algorithms for model checking are based on the use of assertions and tableau rules which are *proof rules*. Assertions are typically of the form $s \vdash_M \phi$ and mean that state s in model M satisfies the formula ϕ. Using a set of tableau rules we aim to prove the truth or falsity of assertions. But unlike traditional proof systems which are bottom-up approaches, tableau-based algorithms work in a *top-down* or *goal-oriented* fashion. Tableau rules are used in order to prove a certain formula by inferring when a state in a Kripke structure satisfies such a formula. According to this approach, we start from a goal, and we apply a proof rule and determine the sub-goals to be proven. The proof rules are designed so that the goal is true if all the sub-goals are true. The advantage of this method is that the state space is explored in a need-driven fashion. The algorithm searches only the part of the state space that needs to be explored to prove or disprove a certain formula.

The tableau rules of CTL^{*CA} are given in Figs. 1–4. We introduce a syntactical operator "?" to express the tableau rule of the challenge action. Syntactically, "?ψ" means that a given agent does not know whether ψ is true or not.

$$R1 \quad \wedge : \frac{\psi_1 \wedge \psi_2}{\psi_1 \; \psi_2} \qquad R2 \quad \vee : \frac{\psi_1 \vee \psi_2}{\psi_1 \; \psi_2} \qquad R3 \quad \vee : \frac{E(\psi)}{\psi}$$

$$R4 \quad \neg : \frac{\neg\psi}{\psi} \qquad R5 \quad ? : \frac{?\psi}{\psi} \qquad R6 \quad \neg : \frac{A(\Phi)}{E(\neg\Phi)}$$

Fig. 1. Tableau rules for propositional and universal formulas.

$$R7 < C >: \frac{E(\Phi, C(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_1, Ag_2, t, \phi))}$$

$$R8 < W >: \frac{E(\Phi, Withdraw(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, \neg PC(Ag_1, Ag_2, t, \phi))}$$

$$R9 < S_{PC}^{Ag1} >: \frac{E(\Phi, Satisfy(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, \phi)}$$

$$R10 < V_{PC}^{Ag1} >: \frac{E(\Phi, Violate(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, \neg\phi)}$$

$$R11 < Rea >: \frac{E(\Phi, Reactivate(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_1, Ag_2, t, \phi))}$$

$$R12 < Ch >: \frac{E(\Phi, Challenge(Ag_2, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_2, Ag_1, t, ?\phi))}$$

$$R13 < Acc >: \frac{E(\Phi, Accept(Ag_2, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_2, Ag_1, t, \phi))}$$

$$R14 < Ref >: \frac{E(\Phi, Refuse(Ag_2, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_2, Ag_1, t, \neg\phi))}$$

$$R15 < Jus >: \frac{E(\Phi, Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))}{E(\Phi, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))}$$

$$R16 < Att >: \frac{E(\Phi, Attack(Ag_2, PC(Ag_1, Ag_2, t, \phi' \therefore \neg\phi)))}{E(\Phi, PC(Ag_2, Ag_1, t, \phi' \therefore \neg\phi))}$$

$$R17 < Def >: \frac{E(\Phi, Defend(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))}{E(\Phi, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))}$$

Fig. 2. Tableau rules for action formulas.

$$R18 \quad [PC_{Ag_1}] : \frac{E(\Phi, PC(Ag_1, Ag_2, t, \phi))}{E(\Phi, \phi)}$$

Fig. 3. Tableau rule for propositional commitment formula.

Tableau rules enable us to define top-down proof systems. The idea is: given a formula, we apply a tableau rule and determine the sub-formulae to be proven. Tableau rules are inference rules used in order to prove a formula by proving all the sub-formulae. The labels of these rules are the labels of states in the automata constructed from a given formula. For example, rule $R1$ of Fig. 1 labeled by "$\wedge$" indicates that ψ_1 and ψ_2 are the two sub-formulae of $\psi_1 \wedge \psi_2$. This means that, in order to prove that a state labeled by "$\wedge$" satisfies the formula $\psi_1 \wedge \psi_2$, we have to prove that the two children of this state satisfy ψ_1 and ψ_2 respectively. According to rule $R2$, in order to prove that a state labeled by "$\vee$" satisfies the formula $\psi_1 \vee \psi_2$, we have to prove that one of the two children of this state satisfies ψ_1 or ψ_2. Rule $R3$ labeled by "$\vee$" indicates that ψ is the sub-formula to be proved in order to prove that a state satisfies $E(\psi)$. According to rule $R4$ (resp. $R5$), the formula $\neg\psi$ (resp. $?\psi$) is satisfied in a state labeled by "$\neg$" (resp. ?), if this state has a successor representing ψ. Rule $R6$ is defined in the usual way where Φ is a set of path formulae.

The label "$\langle C \rangle$" (rule $R7$) is the label associated with the creation action of a propositional commitment PC. According to this rule, in order to prove that a state satisfies $C(Ag_1,PC(Ag_1,Ag_2,t,\phi))$, we have to prove that an accessible state via a transition labeled by the creation action satisfies the sub-formula $PC(Ag_1,Ag_2,t,\phi)$. The rules $R8$–$R17$ are defined in the same way.

Rule $R18$ of Fig. 3 indicates that $E(\phi)$ is the sub-formula of the formula $E(PC(Ag_1,Ag_2,t,\phi))$. Thus, in order to prove that a state satisfies $E(PC(Ag_1,Ag_2,t,\phi))$, we have to prove that the accessible state via a transition labeled by "$[PC_{Ag_1}]$" satisfies $E(\phi)$.

Finally, the rules $R19$ to $R27$ of Fig. 4 are defined in the usual way. For example, according to rule $R24$, in order to prove that a state satisfies $E(X^+\varphi)$, we have to prove that the next state via the transition labeled by "X^+" satisfies the sub-formula $E(\varphi)$.

5. Protocol for communicating agents

5.1. Protocol specification

In this section we define the theoretical model of our model checking procedure. Such a model specifies the dialogue game protocols for agent communication. In this paper we only consider synchronous communications between two agents in the sense that agents alter utterances during the protocol execution. To define this notion formally, let us introduce the following notation: uAg indicating an utterance u made by an agent Ag in a given communication (i.e. protocol execution).

Definition 1. Let Ag_1 and Ag_2 be two communicative agents. A synchronous communication between Ag_1 and Ag_2 is an ordering

$$R19 \quad <\equiv>: \frac{E(\Phi, l)}{l, \; E(\Phi)} \qquad R20 \quad \wedge : \frac{E(\Phi, \phi_1 \wedge \phi_2)}{E(\Phi, \phi_1, \phi_2)} \qquad R21 \quad \vee : \frac{E(\Phi, \phi_1 \vee \phi_2)}{E(\Phi, \phi_1) \; E(\Phi, \phi_2)}$$

$$R22 \quad ? : \frac{E(\Phi, ?\psi)}{E(\Phi, \psi)}$$

$$R23 \quad X^- : \frac{E(\Phi, X^-\phi_1, \ldots, X^-\phi_n)}{E(\Phi, \phi_1, \ldots, \phi_n)} \qquad R24 \quad X^+ : \frac{E(\Phi, X^+\phi_1, \ldots, X^+\phi_n)}{E(\Phi, \phi_1, \ldots, \phi_n)}$$

$$R25 \quad \wedge : \frac{E(\Phi, \phi_1 \therefore \phi_2)}{E(\Phi, \phi_1, X^+(\neg\phi_1 \vee \phi_2)}$$

$$R26 \quad \vee : \frac{E(\Phi, \phi_1 U^- \phi_2)}{E(\Phi, \phi_2) \; E(\Phi, \phi_1, X^-(\phi_1 U^- \phi_2))} \qquad R27 \quad \vee : \frac{E(\Phi, \phi_1 U^+ \phi_2)}{E(\Phi, \phi_2) \; E(\Phi, \phi_1, X^+(\phi_1 U^+ \phi_2))}$$

Fig. 4. Tableau rules for state formulas.

sequence of utterances $u_1; u_2; \ldots; u_n$ such that $\forall i \in [1,n-1], u_iAg_1 \Rightarrow u_{i+1}Ag_2$ and $\forall i \in [1,n-1], u_iAg_2 \Rightarrow u_{i+1}Ag_1$.

Formally, protocols are specified as a set of rules describing the entry condition, the dynamics and the exit condition of the protocol [6]. These rules can be specified in the logic for communicating agents as action formulae (actions on propositional commitments).

We define these protocols as transition systems. The purpose of these transition systems is to describe not only the sequence of the allowed actions (classical transition systems), but also the structure of these actions. The states of these transition systems are sub-transition systems (called *structure transition systems*) describing the structure of the actions labeling the entry transitions. Defining transition systems in such a way allows for the verification of: (1) the correctness of the protocol (if the model of the protocol satisfies the properties that the protocol should specify) and (2) the compliance to the structure of the communicative actions (if the specification of the protocol respects the structure).

The definition of the transition system of dialogue game protocols is given by the following definitions:

Definition 2. A structure transition system T' describing the structure of an action formula is a six-tuple $\langle S', \mathrm{Lab}', F, \mathrm{Ls}', R, \rightarrow, s'_0\rangle$ where:

- S' is a set of states,
- Lab′: $S' \rightarrow 2^{\Phi p}$ is the labeling state function, where Φ_p is the set of atomic propositions,
- F is a sub-set of CTL^{*CA} formulae (F does not include the action formulae, i.e. *Satisfy*, *Accept*, etc.),
- Ls′: $S' \rightarrow F$ is a function associating to each state a formula,
- $R \in \{\wedge, \vee, \neg, ?, \Leftrightarrow, X^+, X^-, PC_{Ag}\}$ is the set of tableau rule labels (without the rules for action formulae),
- $\rightarrow \subseteq S' \times R \times S'$ is the transition relation,
- s'_0 is the start state.

Intuitively, states s' contain the sub-formulae of the action formulae, and the transitions are labeled by operators associated with the formula of the starting state. Semantic transition systems enable us to describe the semantics of formulae by sub-formulae connected by logical operators. Thus, there is a transition between states s'_i and s'_j iff $L'(s'_j)$ is a sub-formula or an semantically equivalent formula of $L'(s'_i)$. Following traditional usage we write $s \rightarrow^r s'$ instead of $\langle s, r, s'\rangle \in \rightarrow$ where $s, s' \in S'$ and $r \in R$.

Definition 3. A transition system T for a dialogue game protocol is a six-tuple $\langle S, \mathrm{Lab}, \wp, \mathrm{L}, \mathrm{Act}, \rightarrow, s_0\rangle$ where:

- S is a set of states,
- Lab: $S \rightarrow 2^{\Phi p}$ is the labeling state function,
- $\wp$ is a set of structure transition systems with $\varepsilon \in \wp$ is the empty semantic transition system,
- L:$S \rightarrow \wp$ is the function associating to a state $s \in S$ a semantic transition system $T' \in \wp$ describing the semantics of the action labeling the entry transition,
- Act $\in$ {C, *Withdraw*, *Satisfy*, *Accept*, *Refuse*, *Challenge*, *Justify*, *Defend*, *Attack*} is the set of actions,
- $\rightarrow \subseteq S \times \mathrm{Act} \times S$ is the transition relation,
- s_0 is the start state with $L(s_0) = \varepsilon$ (i.e. there is no structure transition system in s_0).

The transitions are labeled by the actions applied to propositional commitments. We write $s \rightarrow s'$ instead of $\langle s, \bullet, s'\rangle \in \rightarrow$ where $s, s' \in S$ and $\bullet \in Act$. Fig. 5 illustrates a part of a transition system for a dialogue game protocol.

5.2. Examples of protocol properties

The properties to be verified in the dialogue game protocols specified in CTL^{*CA} are action and temporal properties. For example, we can verify if a model of dialogue game protocol satisfies the following property:

$$AG^+(Challenge(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \Rightarrow F^+\ Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))$$

This property says that in all paths (A), globally (G^+), if an agent Ag_2 challenges the content ϕ of an Ag_1's propositional commitment (PC), then in the future (F^+)Ag_1 will justify this content by an argument $\phi' \therefore \phi$.

Another interesting property to be checked in dialogue games is related to the communicative acts an agent is allowed to perform at a given moment. For example, it is prohibited to attack a commitment content if the addressee did not commit about this content. This property is specified using the past operator F^- as follows:

$$AG^+(Attack(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \Rightarrow F^-C(Ag_1, PC(Ag_1, Ag_2, t, \phi)))$$

A third property capturing the deontic notion of propositional commitments is given by the following formula:

$$\begin{aligned}&AG^+(Attack(Ag_2, PC(Ag_1, Ag_2, t, \phi' \therefore \neg\phi)) \Rightarrow\\&(F^+Defend(Ag_1, PC(Ag_1, Ag_2, t, \phi'' \therefore \phi))\\&\vee F^+Attack(Ag_1, PC(Ag_2, Ag_1, t', \phi'' \therefore \neg\phi'))\\&\vee F^+Accept(Ag_1, PC(Ag_2, Ag_1, t', \phi'))))\end{aligned}$$

Using this property, we can verify if a model of a dialogue game protocol satisfies the fact that if an agent Ag_2 attacks the content of an agent Ag_1's propositional commitment PC, then Ag_1 will defend its propositional commitment content, attack the Ag_2's argument or accept it.

6. Model checking technique

In this section, we use a combination of an automata-theoretic approach and a tableau-based approach to model checking communicating agent-based systems.

6.1. Alternating Büchi tableau automata for CTL$^{*\ CA}$

As a kind of Büchi automata, ABTAs [9] are used in order to prove properties of *infinite* behavior. These automata can be used as an intermediate representation for system properties. Let Φ_p be the set of atomic propositions and let $\Re$ be a set of tableau rule labels defined as follows: $\Re = \{\wedge, \vee, \neg, ?\} \cup \Re_{Act} \cup \Re_{\neg Act} \cup \Re_{SC} \cup \Re_{Set}$ where $\Re_{Act}, \Re_{SC}$ and $\Re_{Set}$ are defined as follows:

$$\Re_{Act} = \{\langle C\rangle, \langle W\rangle, \langle S^{Ag}_{PC}\rangle, \langle V^{Ag}_{PC}\rangle, \langle Rea\rangle, \langle Ch\rangle, \langle Acc\rangle, \langle Ref\rangle, \langle Jus\rangle, \langle Att\rangle, \langle Def\rangle\}.$$
$$\Re_{SC} = \{[PC_{Ag}]\}.$$
$$\Re_{Set} = \{\iff, X^+, X^-\}.$$

The associated tableau rules are given in Figs. 1–4.

Formally, we define ABTAs for CTL^{*CA} logic as follows:

Definition 4. An ABTA for CTL^{*CA} is a five-tuple $\langle Q, l \rightarrow q_0, F\rangle$, where:

- Q is a finite set of states,
- $l : Q \rightarrow \Phi_p \cup \Re$ *is the state labeling,*
- $\rightarrow \subseteq Q \times Q$ is the transition relation,
- q_0 is the start state,
- $F \subseteq 2^Q$ is the acceptance condition.

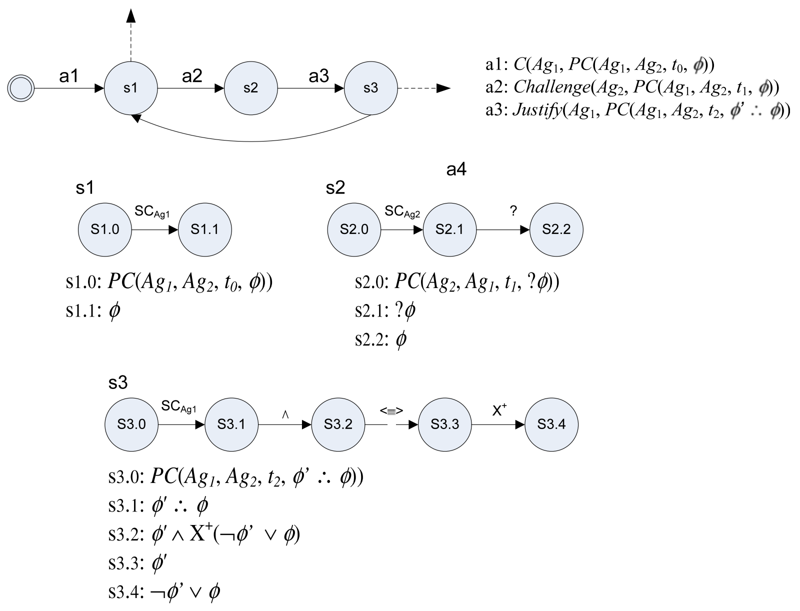

a1
a2
a3
s1
s2
s3
a1: $C(Ag_1, PC(Ag_1, Ag_2, t_0, \phi))$
a2: $Challenge(Ag_2, PC(Ag_1, Ag_2, t_1, \phi))$
a3: $Justify(Ag_1, PC(Ag_1, Ag_2, t_2, \phi' \therefore \phi))$
a4
S1.0
SC_{Ag1}
S1.1
S2.0
SC_{Ag2}
S2.1
?
S2.2
s1.0: $PC(Ag_1, Ag_2, t_0, \phi))$
s1.1: ϕ
s2.0: $PC(Ag_2, Ag_1, t_1, ?\phi))$
s2.1: $?\phi$
s2.2: ϕ
S3.0
SC_{Ag1}
S3.1
$\wedge$
S3.2
<=>
S3.3
X^+
S3.4
s3.0: $PC(Ag_1, Ag_2, t_2, \phi' \therefore \phi))$
s3.1: $\phi' \therefore \phi$
s3.2: $\phi' \wedge X^+(\neg\phi' \vee \phi)$
s3.3: ϕ'
s3.4: $\neg\phi' \vee \phi$

Fig. 5. A part of atransition system for a dialogue game protocol.

ABTAs allow us to encode "*top-down proofs*" for temporal formulae. Indeed, an ABTA encodes a proof schema in order to prove, in a goal-directed manner, that a transition system satisfies a temporal formula.

Example 1. Let us consider the following example. We would like to prove that a state s in a transition system satisfies a temporal formula of the form $F1 \wedge F2$, where $F1$ and $F2$ 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 $F1$, and the second would be to prove that s satisfies $F2$. Intuitively, an ABTA for $F1 \wedge F2$ would encode this "proof structure" using states for the formulae $F1 \wedge F2$, $F1$, and $F2$. A transition from $F1 \wedge F2$ to each of $F1$ and $F2$ should be added to the ABTA and the labeling of the state for $F1 \wedge F2$ being "$\wedge$" which is the label of a certain rule. 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, and (3) the transition relation represents a "sub-goal" relationship.

In order to decide about the satisfaction of formulae, we use the notion of the *accepting runs* of an ABTA on a transition system. These runs are not considered to be finite, but rather infinite, while cycling infinitely many times through acceptance states. In order to define this notion of the ABTA's run, we need to introduce three types of nodes: *positive*, *negative* and *neutral* (neither positive nor negative). Intuitively, nodes classified positive are nodes that correspond to a formula without negation (for example $C(Ag_1,PC(Ag_1,Ag_2,t,\phi)))$, and negative nodes are nodes that correspond to a formula with negation (for example $\neg Justify(Ag_1,PC(Ag_1,Ag_2,t,\phi'\phi)))$. Neutral nodes are used in order to verify the semantics of an action formula ($act \in Act$) written in the formula to be verified under the form $\neg act$. From the syntax point of view, $\neg act$ means that the action *act* is not performed. For example, if in the formula to be verified appears the sub-formula: $\neg Justify(Ag_1,PC(Ag_1,Ag_2,t,\phi'\phi))$, we use in the ABTA neutral nodes in order to verify the semantics of: $Justify(Ag_1,PC(Ag_1,Ag_2,t,\phi'\phi))$.The reason is that in transition systems, and consequently in the sub-transition systems, we have only action formulae without negation, whereas in the formula to be verified, we can have action formulae with negation. We note that we cannot use here negative nodes because we do not interested in the formula in itself (i.e. in the example $\neg Justify(Ag_1,PC(Ag_1,Ag_2,t,\phi'\phi)))$ but in the semantics of the underlying action (i.e. $Justify(Ag_1,PC(Ag_1,Ag_2,t,\phi'\phi)))$. In other words, we are not interested in the semantics of the negation action, but in the semantics of the action itself. We note here that in order to verify that an action formula $\neg act$ is satisfied, we have to verify that from a given state there is no transition in the transition system labeled by *act*. Definition 5 gives the definition of this notion of run. In this definition, elements of the set S of states are denoted s_i or t_i. The explanation of the different clauses is given after the definition.

Definition 5. A run of an ABTA $B = \langle Q, l, \rightarrow, q_o, F\rangle$ on a transition system $T = \langle S, \text{Lab}, \wp, L, \text{Act}, \rightarrow, s_0\rangle$ is a graph in which the nodes are classified as positive, negative or neutral and are labeled by elements of $Q \times S$ as follows:

1. The root of the graph is a positive node and is labeled by $\langle q_0, s_0\rangle$.

2. If φ is a positive node with label $\langle q, s_i\rangle$ such that $l(q) = \neg$ and $q \rightarrow q'$, then φ has one negative successor labeled $\langle q', s_i\rangle$ and vice versa.
   - Otherwise, for a positive node φ labeled by $\langle q, s_i\rangle$:

3. If $l(q) \in \Phi_p$ then φ is a leaf.

4. If $l(q) \in \{\wedge, \Leftrightarrow\}$ and $\{q' |\ q \rightarrow q'\} = \{q_1, \ldots, q_m\}$, then φ has positive successors $\varphi_1, \ldots, \varphi_m$ with φ_j labeled by $\langle q_j, s_i\rangle$ $(1 \leqslant j \leqslant m)$.

5. If $l(q) = \vee$ then φ has one positive successor φ' labeled by $\langle q', s_i\rangle$ for some $q' \in \{q' | q \rightarrow q'\}$.

6. If $l(q) = X^+$ and $q \rightarrow q'$ and $\{s' | s_i \rightarrow^{\bullet} s'\} = \{t_1, \ldots, t_m\}$, where $\bullet \in \text{Act}$, then φ has positive successors $\varphi_1, \ldots, \varphi_m$ with φ_j labeled by $\langle q', t_j\rangle$ $(1 \leqslant j \leqslant m)$.

7. If $l(q) = X^-$ and $q \rightarrow q'$ and $\{s' | s' \rightarrow^{\bullet}\ s_i\} = \{t_1, \ldots, t_m\}$, where $\bullet \in \text{Act}$, then φ has positive successors $\varphi_1, \ldots, \varphi_m$ with φ_j labeled by $\langle q', t_j\rangle$ $(1 \leqslant j \leqslant m)$.

8. If $l(q) = \langle\bullet\rangle$, where $\bullet \in \text{Act}$ and $q \rightarrow q'$, and $s_i \rightarrow^{\bullet} s_{i+1}$ then φ has one positive successor φ' labeled by $\langle q', s_{i+1,0}\rangle$ where $s_{i+1,0}$ is the initial state of the semantic transition system of s_{i+1}.

9. If $l(q) = \langle \bullet \rangle$, where $\bullet \in \neg$ Act and $q \to q'$, and $s_i \to^{\neg\bullet} s_{i+1}$ then φ has one neutral successor φ' labeled by $\langle q', s_{i+1,0} \rangle$ where $s_{i+1,0}$ is the initial state of the semantic transition system of s_{i+1}.

10. If $l(q) = \langle \bullet \rangle$, where $\bullet \in \neg$Act and $q \to q'$, and $s_i \to^{\bullet'} s_{i+1}$ where $\bullet \neq \bullet'$ and $\bullet' \in$ Act, then φ has one positive successor φ' labeled by $\langle q', s_{i+1} \rangle$.

- Otherwise, for a negative node φ labeled by $\langle q, s_i \rangle$:

11. If $l(q) \in \Phi_p$ then φ is a leaf.

12. If $l(q) \in \{\vee, \Leftrightarrow\}$ and $\{q' | q \to q'\} = \{q_1, \ldots, q_m\}$, then φ has negative successors $\varphi_1, \ldots, \varphi_m$ with φ_j labeled by $\langle q_j, s_i \rangle$ $(1 \leqslant j \leqslant m)$.

13. If $l(q) = \wedge$ then φ has one negative successor φ' labeled by $\langle q', s_i \rangle$ for some $q' \in \{q' | q \to q'\}$.

14. If $l(q) = X^+$ and $q \to q'$ and $\{s' | s_i \to^{\bullet} s'\} = \{t_1, \ldots, t_m\}$ where $\bullet \in Act$, then φ has negative successors $\varphi_1, \ldots, \varphi$m with φ_j labeled by $\langle q', t_j \rangle$ $(1 \leqslant j \leqslant m)$.

15. If $l(q) = X^-$ and $q \to q'$ and $\{s' | s' \to^{\bullet} s_i\} = \{t_1, \ldots, t_m\}$, where $\bullet \in Act$, then φ has negative successors $\varphi_1, \ldots, \varphi_m$ with φ_j labeled by $\langle q', t_j \rangle$ $(1 \leqslant j \leqslant m)$.

16. If $l(q) = \langle \bullet \rangle$, where $\bullet \in$ Act and $q \to q'$, and $s_i \to^{\bullet} s_{i+1}$ then φ has one negative successor φ' labeled by $\langle q', s_{i+1,0} \rangle$ where $s_{i+1,0}$ is the initial state of the semantic transition system of s_{i+1}.

17. If $l(q) = \langle \bullet \rangle$, where $\bullet \in \neg$Act and $q \to q'$, and $s_i \to^{\neg\bullet} s_{i+1}$ then φ has one neutral successor φ' labeled by $\langle q', s_{i+1,0} \rangle$ where $s_{i+1,0}$ is the initial state of the semantic transition system of s_{i+1}.

18. If $l(q) = \langle \bullet \rangle$, where $\bullet \in \neg$Act and $q \to q'$, and $s_i \to^{\bullet'} s_{i+1}$ where $\bullet \neq \bullet'$ and $\bullet' \in$ Act, then φ has one negative successor φ' labeled by $\langle q', s_{i+1} \rangle$.

- Otherwise, for a neutral node φ labeled by $\langle q, s_{i,j} \rangle$:

19. If $l(q) = \Leftrightarrow$ and $\{q' | q \to q'\} = \{q_1, q_2\}$ such that q_1 is a leaf, and $s_{i,j}$ has a successor $s_{i,j+1}$, then φ has one positive leaf successor φ' labeled by $\langle q_1, s_{i,j} \rangle$ and one neutral successor φ'' labeled by $\langle q_2, s_{i,j+1} \rangle$.

20. If $l(q) = \Leftrightarrow$ and $\{q' | q \to q'\} = \{q_1, q_2\}$ such that q_1 is a leaf, and $s_{i,j}$ has no successor, then φ has one positive leaf successor labeled by $\langle q_1, s_{i,j} \rangle$.

- Otherwise, for a positive (negative) node φ labeled by $\langle q, s_{i,j} \rangle$:

21. If $l(q) = \Leftrightarrow$ and $\{q' | q \to q'\} = \{q_1, q_2\}$ such that q_1 is a leaf, and $s_{i,j}$ has a successor $s_{i,j+1}$, then φ has one positive leaf successor φ' labeled by $\langle q_1, s_{i,j} \rangle$ and one positive (negative) successor φ'' labeled by $\langle q_2, s_{i,j+1} \rangle$.

22. If $l(q) = \Leftrightarrow$ and $\{q' | q \to q'\} = \{q_1, q_2\}$ such that q_1 is a leaf, and $s_{i,j}$ has no successor, then φ has one positive leaf successorφ' labeled by $\langle q_1, s_{i,j} \rangle$ and one positive (negative) successor φ'' labeled by $\langle q_2, s_i \rangle$.

- Otherwise, for a positive (negative, neutral) node φ labeled by $\langle q, s_{i,j} \rangle$:

23. If $l(q) \in \{\wedge, \vee, ?, X^+, X^-, [PC_{Ag}]\}$ and $\{q' | q \to q'\} = \{q_1\}$, and $s_{i,j} \to^{r} s_{i,j+1}$ such that $r = l(q)$, then φ has one positive (negative, neutral) successor φ' labeled by $\langle q_1, s_{i,j+1} \rangle$.

The notion of run of an ABTA on a transition system is a non-synchronized product graph of the ABTA and the transition system. This run uses the label of nodes in the ABTA ($l(q)$), the transitions in the ABTA ($q \to q'$), and the transitions in the transition system ($s_i \to s_j$). The product is not synchronized in the sense that it is possible to use transitions in the ABTA while staying in the same state in the transition system (this is the case for example of the clauses 2, 4, and 5).

The second clause in the definition says that if we have a positive node φ in the product graph such that the corresponding state in the ABTA is labeled with $\neg$and we have a transition $q \to q'$ in this ABTA, then φ has one negative successor labeled with $\langle q', s_i \rangle$. In this case we use a transition from the ABTA and we stay in the same state of the transition system. In the case of a positive node and if the current state of the ABTA is labeled with $\wedge$, all the transitions of this current state of the ABTA are used (clause 4). However, if the current state of the ABTA is labeled with $\vee$, only one arbitrary transition from the ABTA is used (clause 5). The intuitive idea is that in the case of $\wedge$, all the sub-formulae must be true in order to decide about the formula of the current node of the ABTA, and in the case of $\vee$ only one sub-formula must be true.

The cases in which a transition of the transition system is used are:

1. The current node of the ABTA is labeled with X^+ (which means a next state in the transition system) or X^- (which means a previous state in the transition system). This is the case of the clauses 6, 7, 14, and 15. In this case we use all the transitions from the current state s_i to next or previous states of the transition system.
2. The current state of the ABTA and a transition from the current state of the transition system are labeled with the same action. This is the case of the clauses 8 and 16. In this case, the current transition of the ABTA and the transition from the current state s_i of the transition system to a state $s_{i+1,0}$ of the associated semantic transition system are used. The idea is to start the parsing of the formula coded in the semantic transition system.
3. The current state of the ABTA and a transition from the current state of the transition system are labeled with the same action which is preceded by $\neg$ in the ABTA. This is the case of the clauses 9 and 17. In this case, the current transition of the ABTA and the transition from the current state s_i of the transition system to a state $s_{i+1,0}$ of the associated semantic transition system are used. The successor node is classified neutral. This allows us to verify the structure of the formula coded in the transition system.
4. The current state of the ABTA and a transition from the current state of the transition system are labeled with different actions where the state of the ABTA is labeled with a negative formula. This is the case of the clauses 10 and 18. In this case, the formula is satisfied, but its structure cannot be verified. Consequently, the current transition of the ABTA and the transition from the current state s_i of the transition system to a next state s_{i+1} are used. This means that, we do not visit the associated semantic transition system.

Finally, the clauses 19, 20, 21, 22, and 23 deal with the case of verifying the structure of the commitment formulae in the sub-transition systems. In these clauses, transitions $s_{i,j} \rightarrow s_{i,j+1}$ are used. We note here that when $s_{i,j}$ has no successor, the formula contained in this state is an atomic formula or a boolean formula whose all the sub-formulae are atomic (for example $p \wedge q$ where p and q are atomic).

We also need to define the notion of *success* of a run for the correctness of the model checking. To define this notion, we first introduce *positive* and *negative paths*. In an ABTA, every infinite path has a suffix that contains either positive or negative nodes, but not both. Such a path is referred to as *positive* in the former case and *negative* in the latter.

Let $p \in \Phi p$ and let s_i be a state in a transition system T. Then $s_i \vDash_T p$ iff $p \in Lab(s_i)$ and $s_i \vDash_T \neg p$ iff $p \notin Lab(s_i)$.

Let $s_{i,j}$ be a state in a semantic transition system of a transition system T. Then $s_{i,j} \vDash_T p$ iff $p \in Lab'(s_{i,j})$ and $s_{i,j} \vDash_T \neg p$ iff $p \notin Lab'(s_{i,j})$.

Definition 6. Let r be a run of ABTA $B = \langle Q, l, \rightarrow, q_0, F\rangle$ on a transition system $T = \langle S, Lab, \wp, L, Act, \rightarrow, s_0\rangle$. The run r is successful iff every leaf and every infinite path in r is successful. A successful leaf is defined as follows:

1. A positive leaf labeled by $\langle q, s_i\rangle$ is successful iff $s_i \vDash_T l(q)$ or $l(q) = \langle \bullet \rangle$ where $\bullet \in$ Act and there is no s_j such that $s_i \rightarrow {}^{\bullet}s_j$.
2. A positive leaf labeled by $\langle q, s_{i,j}\rangle$ is successful iff $s_{i,j} \vDash_T l(q)$
3. A negative leaf labeled by $\langle q, s_i\rangle$ is successful iff $s_i \vDash_T \neg l(q)$ or $l(q) = \langle \bullet \rangle$ where $\bullet \in$ Act and there is no s_j such that $s_i \rightarrow {}^{\bullet}s_j$.
4. A negative leaf labeled by $\langle q, s_{i,j}\rangle$ is successful iff $s_{i,j} \vDash_T \neg l(q)$
5. All neutral leaves are not successful.

A successful infinite path is defined as follows:

1. A positive path is successful iff $\forall f \in F, \exists q \in f$ such that q occurs infinitely often in the path. This condition is called the Büchi condition.
2. A negative path is successful iff $\exists f \in F, \forall q \in f$, q does not occur infinitely often in the path. This condition is called the co-Büchi condition.

We note here that a positive or negative leaf labeled by $\langle q, s\rangle$ such that $l(q) = \langle \bullet \rangle$ where $\bullet \in Act$ and there is no s' such that $s \rightarrow {}^{\bullet}s'$ is considered a successful leaf because we cannot consider it unsuccessful. The reason is that it is possible to find a transition labeled by $\bullet$ and starting from another state s'' in the transition system. This is the case of the leaf labeled by $(\langle Ch\rangle, s_0)$ in the *case study* we will discuss in Section 8 (see Fig. 11). If we consider such a leaf unsuccessful, then even if we find a successful infinite path, the run will be considered unsuccessful. However this is false. We also note that an ABTA B accepts a transition system T iff there exists a successful run of B on T.

6.2. Translation procedure

The procedure for translating a CTL^{*CA} formula $p = E\phi$ to an ABTA B uses goal-directed rules in order to build a tableau from this formula. Indeed, these proof rules are conducted in a top-down fashion in order to determine whether states satisfy properties or not. The tableau is constructed by exhaustively applying the rules contained in Figs. 1–4 to p. Then, B can be extracted from this tableau as follows. First, we generate the states and the transitions. Intuitively, states will correspond to state formulae, with the start state being p. To generate new states from an existing state for a formula p', we determine which rule is applicable to p', starting with $R1$, by comparing the form of p' to the formula appearing in the "goal position" of each rule. Let *rule*(q) denote the rule applied at node q. The labeling function l of states is defined as follows. If q does not have any successor, then $l(q) \in \Phi_p$. Otherwise, the successors of q are given by *rule*(q). The label of the rule becomes the label of the state q, and the sub-goals of the rule are then added as states related to q by transitions.

A tableau for a CTL^{*CA} formula p is a maximal proof tree having p as its root and constructed using rules $R1$–$R27$. If p' results from the application of a rule to p, then we say that p' is a child of p in the tableau. The height of a tableau is defined as the length of the longest sequence $\langle p_0, p_1, \ldots\rangle$, where p_{i+1} is the child of p_i [13]. Finally, in order to compute the successful run of the generating ABTA, we should compute the acceptance states F. For this purpose we use the following definition.

Definition 7. Let q be a state in an ABTA B and Q the set of all states. Suppose $\phi = \phi_1 U^{+} \phi_2 \in q$.[1] We define the set F_ϕ as follows:

$$F_\phi = \{q' \in Q \mid (\phi \notin q' \text{ and } X^{+}\phi \notin q') \text{ or } \phi_2 \in q'\}.$$

The acceptance set F is defined as follows :

$$F = \{F_\phi \mid \phi = \phi_1 U^{+} \phi_2 \text{ and } \exists q \in B, \phi \in q\}.$$

According to this definition, a state that contains the formula ϕ or the formula $X^{+}\phi$ is not an acceptance state. The reason is that according to Definition 5, there is a transition from a state containing ϕ to a state containing $X^{+}\phi$ and vice versa. Therefore, according to Definition 6, there is a successful run in the ABTA B. However, we cannot decide about the satisfaction of a formula using this run. The reason is that in an infinite cycle including a state containing ϕ and a state containing $X^{+}\phi$, we cannot be sure that a state containing ϕ_2 is reachable. However, according to the semantics of U^{+}, the satisfaction of ϕ needs that a state containing ϕ_2 is reachable while passing by states containing ϕ_1. To illustrate the practical issues about the translation procedure, we consider two examples with different formulas (Examples 2 and 3). Furthermore, a complete example with more practical considerations is presented and discussed in Section 8 (the case study).

Example 2. Let us show a practical case on how a CTL^{*CA} formula is translated to an ABTA. We consider the following propositional formula: $E(G^{+}F^{+}p)$. In the context of dialogue game-based agents, this formula says that along some transitions, globally in the future a commitment content holds. The first step is to build the tableau for this formula using tableau rules. The first rule we can apply is $R27$ labeled by "$\vee$" for the until formula (G^{+} is an abbreviation defined from U^{+}). The second rule is also $R27$ for $F^{+}p$ (F^{+} is also an abbreviation defined from U^{+}). Thereafter rules $R19$ and $R24$ can be applied. We obtain the tableau illustrated in Fig. 6 where the rule labels are indicated.

The ABTA obtained from this tableau is illustrated in Fig. 7. In this ABTA, states (1), (3), (5), and (6) are the acceptance states according to Definition 7. The formula ϕ we consider is the following: $\phi = \text{true}U^{+}p \equiv F^{+}p$. Notice that ϕ and $X^{+}\phi$ do not appear in these states. State (5) is the acceptance state in the finite case. On the other hand, ϕ appears in states (2) and (7), and $X^{+}\phi$ appears in state (4). Therefore, these states are not in F_ϕ. The path $\Pi = (1,(2,4,7)^{*})$ is not a valid proof of $E(G^{+}F^{+}p)$. However, a path that visits infinitely often the states (1), (3), and (6) is a valid (infinite) proof. The reason is that in such a path there is always a chance to meet the proposition p (state (3)). Therefore, this path satisfies the Büchi condition. The Büchi condition is not satisfied in the path Π since there is no chance to visit infinitely often a state containing p.

[1] Here we consider "until" formula because is the formula that allows paths to be infinite.

$$\vee : E(G^+F^+p)\ (1)$$

$$\vee : E(F^+p, X^+G^+F^+p)\ (2)$$

$$<\equiv>: E(p, X^+G^+F^+p)\ (3) \qquad X^+ : E(X^+F^+p, X^+G^+F^+p)\ (4)$$

$$p\ (5) \quad X^+ : E(X^+G^+F^+p)\ (6) \qquad \vee : E(F^+p, G^+F^+p)\ (7)$$

$$E(G^+F^+p) \qquad E(F^+p, X^+G^+F^+p)$$

Fig. 6. The tableau for $E(G^+F^+p^+)$.

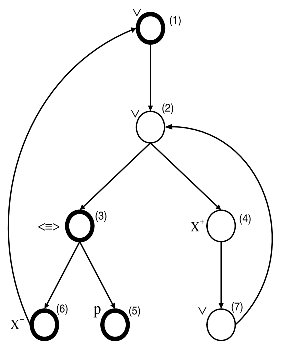


Fig. 7. The ABTA of the formula $E(G^+F^+p)$ using the translation procedure.

Example 3. Let us now consider another practical example illustrating a more complicated formula. This example shows more practical issues on the translation procedure. The formula we study here is: $A(F^+G^+\ Acc(Ag_2,PC(Ag_1,Ag_2,t,\phi)))$ where *Acc* is an abbreviation of *Accept*. This CTL^{*CA} formula says that in all paths, in some future along these paths, in all states agent Ag_2 will accept the commitment done by Ag_1. As for the previous example, the first step is to build the tableau for this formula using tableau rules. Fig. 8 depicts the resulting tableau tree. The first applicable rule is R6 labeled by "¬", which allows us to perform the transition from the universal to the existential quantifiers. According to this rule, the conclusion should be the negation of Acc, which is the refusal action (*Ref* is the abbreviation of *Refuse*). The second rule we can apply is *R*27 labeled by "∨" for the until formula. The third rule is also *R*27 for F^+Ref. Rules 4 (*R*14) and 6 (*R*18) are the rules used for action formulas since refusal and propositional commitments are successively obtained. Rule 5 (*R*24) is the applicable rule since the next operator is obtained after rule 3. The rest of the rules are straightforward. We obtain the tableau illustrated in Fig. 8.

The ABTA obtained from this tableau is depicted in Fig. 9. From the formula we consider in this example, we obtain the following accepting formula used to compute the acceptance states: $F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi))$. According to Definition 7, the acceptance states of this example are: (1), (2), (4), (6), (8), (9), (10), and (11). Notice that ϕ and $X^+\phi$ do not appear in these states. States (9) and (10) are the acceptance states in the finite case. On the other hand, ϕ and $X^+\phi$ appear in states (3), (5), and (7). Therefore, these states are not in F_ϕ. The path $\Pi = (1,2,(3,5,7)^*)$ is not a valid proof of the considered formula. However, a path that visits infinitely often the non-finale acceptance states is a valid (infinite) proof. The

$$\neg : A(F^+G^+Acc(Ag_2,PC(Ag_1,Ag_2,t,\phi)))\ (1)$$

$$\vee : E(G^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))\ (2)$$

$$\vee : E(F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)),\ X^+G^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))\ (3)$$

$$<Ref>: E(Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)), X^+G^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))\ (4) \qquad X^+ : E(X^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)), X^+G^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))\ (5)$$

$$[PC_{Ag_1}] : E(PC(Ag_1,Ag_2,t,\neg\phi), X^+G^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))\ (6) \qquad \vee : E(F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)), G^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))\ (7)$$

$$<\equiv>: E(\neg\phi, X^+G^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))\ (8) \qquad E(F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)), X^+G^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))$$

$$\neg : \neg\phi\ (9) \qquad X^+ : E(X^+G^+F^+ Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))\ (10)$$

$$\phi\ (11) \qquad E(G^+F^+Ref(Ag_2,PC(Ag_1,Ag_2,t,\phi)))$$

Fig. 8. The tableau for $A(F^+G^+Acc(Ag_2,PC(Ag_1,Ag_2,t,\phi)))$.

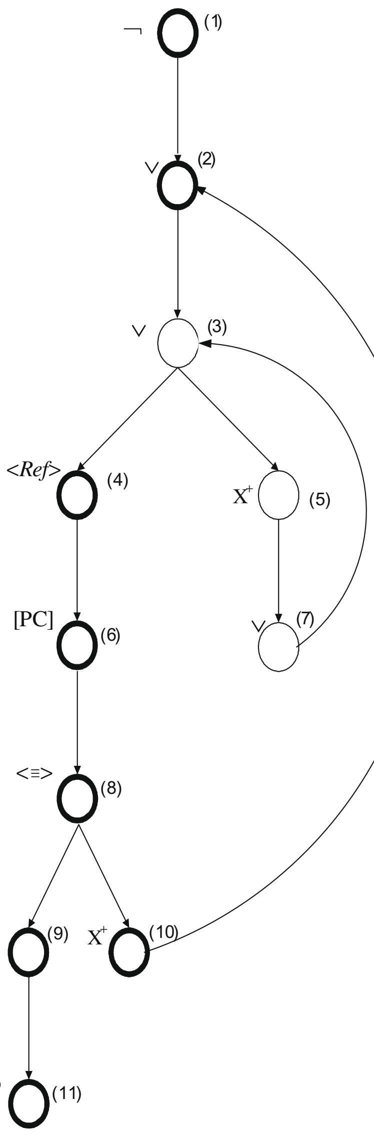


Fig. 9. The ABTA of the formula $A(F^{+}G^{+}\ Acc(Ag_2,PC(Ag_1,Ag_2,t,\phi)))$ using the translation procedure.

reason is that in such a path there is always a chance to meet the acceptance action because it is false to have globally in the future the refusal action (we notice that the first state is labeled by ¬). Therefore, this path satisfies the Büchi condition.

As illustrated by these two examples, the formula to be checked is translated in an ABT automaton. The size of the automaton (number of states and transitions) depends on the size of the formula to be checked (number of atomic propositions and connectors). Consequently, for large formulas, the size of the automaton becomes huge, and can explode. However, many solutions can be used to address this issue. The first solution is to divide the large formulas into smaller ones, and then check the formulas individually. This partial check is practically suitable and is one of the advantages of using model checking-based verification. In our technique, an additional, more efficient, solution is adapted. The solution is provided by the efficient technique used in our approach, which is the on-the-fly procedure. According to this procedure, there is no need to build the whole model, but only a relevant part of it. The relevant part is decided by the algorithm in terms of the part we need to check in the system. The other parts are simply ignored. This technique allows us to check the formulas locally. More details about the functioning of this technique are provided in the next section (Section 6.3).

6.3. Model checking algorithm

The idea behind our model checking algorithm is to explore the product graph of an ABTA for CLT^{*CA} and a transition system for a dialogue game. This algorithm is on-the-fly (or *local*) algorithm consisting of checking if a transition system is accepted by an ABTA. This model checking is reduced to the emptiness of the Büchi automata [26].

Let $T = \langle S, Lab, \wp, L, Act, \rightarrow, s_0\rangle$ be a transition system for a dialogue game and let $B = \langle Q, l, \rightarrow, q_0, F\rangle$ be an ABTA for CTL^{*CA}. The procedure consists of building the ABTA product $B_{\otimes}$ of T and B while checking if there is a successful run in $B_{\otimes}$. The existence of such a run means that the language of $B_{\otimes}$ is non-empty. The automaton $B_{\otimes}$ is defined as follows: $B_{\otimes} = \langle Q \times S, \rightarrow_{B\otimes}, q_{0B\otimes}, F_{B\otimes}\rangle$. There is a transition between two nodes $\langle q,s\rangle$ and $\langle q',s'\rangle$ iff there is a transition between these two nodes in some run of B on T. Intuitively, $B_{\otimes}$ simulates all the runs of the ABTA. The set of accepting states $F_{B\otimes}$ is defined as follows: $q_{0B\otimes} \in F_{B\otimes}$ iff $q \in F$.

Unlike the algorithms proposed in [8,9,14], our algorithm uses only one depth-first search (DFS) instead of two. This is due to the fact that our algorithm explores directly the product graph using the sign of the nodes (positive, negative, or neutral). In addition, unlike the algorithm proposed in [9], our algorithm does not distinguish between recursive and non-recursive nodes. Therefore, we do not take into account the strongly-connected components in the ABTA, but we use a marking algorithm that works on the product graph.

The pseudo-code of this algorithm is given in Fig. 10. The idea is to construct the product graph while exploring it. However, in order to make it easy to understand, we omit the instructions relative to the addition of nodes in the product graph. The construction procedure is directly obtained from Definition 5. The algorithm uses the label of nodes in the ABTA, and the transitions in the product graph obtained from the transition system and the ABTA as explained in Definition 5.

In order to decide if the ABTA contains an infinite successful run, all the explored nodes are marked "visited". Thus, when the algorithm explores a visited node, it returns false if the infinite path is not successful. If the node is not already visited, the algorithm tests if it is a leaf. In this case, it returns false if the node is a non-successful leaf. If the explored node is not a leaf, the algorithm calls recursively the function DFS in order to explore the successors of this node. If this node is labeled by "∧", and signed neutrally or positively, then DFS returns false if one of the successors is false. However, if the node is signed negatively, DFS returns false if all the successors are false. A dual treatment is applied when the node is labeled by "∨". We note that if the DFS does not explore a false node (i.e. it does not return false), then it returns true.

Theorem 1. (**algorithm's correctness**) *Let B an ABTA and T a transition system. DFS (q_0, s_0) returns true if and only if T is accepted by B.*

Theorem 2. (**technique's soundness and completeness**) *Let ψ be a CTL^{*CA} formula and B_{ψ} the ABTA obtained by the translation procedure, and let T be a transition system that represents a dialogue game protocol. Then $s_{0'} \models \psi$ iff T is accepted by B_{ψ}.*

The proofs of these theorems are developed in [3].[2]

[2] The proof is developed in Chapter 8 and could be checked online: http://users.encs.concordia.ca/~bentahar/Thesis/BNJ-Thesis1.pdf

```
DFS(v = (q, s)): boolean {
    if v marked visited {
        if (sign(v) = "+" and not accepting(v)) or (sign(v) = "-" and accepting(v))
            return false
    } // end of if v marked visited
    else {
        mark v visited
        switch(l(q)) {
            case ($p \in \Phi p$):
                switch(sign(v)) {
                    case("+"): if s is a sub-state and $l(q) \notin L'(s)$ return false
                    case("-"): if s is a sub-state and $\neg l(q)) \notin L'(s)$ return false
                    case("neutral"): return false
                } // end of switch(sign(v))
            case($\wedge$):
                if s is a leaf return false
                else
                    switch(sign(v)) {
                        case(neutral): for all $v'' \in \{v' / v \rightarrow_{B\otimes} v'\}$
                                           if not DFS(v'') return false
                        case("+"): for all $v'' \in \{v' / v \rightarrow_{B\otimes} v'\}$
                                       if not DFS(v'') return false
                        case("-"): for all $v'' \in \{v' / v \rightarrow_{B\otimes} v'\}$
                                       if DFS(v'') return true else return false
                    } // end of switch(sign (v))
            case($\vee$):
                if s is a leaf return false
                else
                    switch(sign(v)) {
                        case(neutral): for all $v'' \in \{v' / v \rightarrow_{B\otimes} v'\}$
                                           if DFS(v'') return true else return false
                        case("+"): for all $v'' \in \{v' / v \rightarrow_{B\otimes} v'\}$
                                       if DFS(v'') return true else return false
                        case("-"): for all $v'' \in \{v' / v \rightarrow_{B\otimes} v'\}$
                                       if not DFS(v'') return false
                    } // end of switch(sign (v))
            case(<•>):
                if s is a leaf return true
                else for the $v'' \in \{v' / v \rightarrow_{B\otimes} v'\}$ if not DFS(v'') return false
            case($X^+$, $PC_{Ag}$, $AC_{Ag}$, <≡>, ?):
                if s is a leaf return false
                else for the $v'' \in \{v' / v \rightarrow_{B\otimes} v'\}$ if not DFS(v'') return false
        } // end of switch(l(q))
    } // end of else
    return true }
```

Fig. 10. Model checking algorithm.

7. Termination and computational complexity

In this section we prove the termination of the translation procedure and we discuss the worst-case time complexity of our model checking. Since the translation procedure is based on tableau rules, we need to prove the finiteness of the tableau. The methodology we follow is inspired by [1,13].

If σ_2 is a CTL^{*CA} formula resulting from the application of a rule to a CTL^{*CA} formula σ_1, then we say that σ_2 is a child of σ_1 in the tableau and σ_1 is the parent of σ_2. The *height* of a tableau [13] is defined as the length of the longest sequence $\langle \sigma_0, \sigma_1, \ldots \rangle$, where σ_i is the parent of σ_{i+1}. To prove the finiteness of a tableau, we will establish that each formula has a maximum height tableau.

Intuitively, to show the finiteness of the tableau, we will define a strict ordering relation $\prec$ between CTL^{*CA} formulae and then show that: (1) if σ_1 is the parent of σ_2, then $\sigma_1 \prec \sigma_2$ and (2) the strict ordering relation $\prec$ has no infinite ascending chains.

The ordering relation $\prec$ should reflect the fact that applying tableau rules results in shorter formulae or recursive formulae. The idea is to prove that the number of nodes of the ABTA is finite. Therefore, the definition of this ordering is based either on the fact that formulae are recursive or on the length of formulae. We notice that in the case of recursive formulae, we obtain cycles which are infinite paths on a finite number of nodes. The length of a formula is defined inductively as follows:

Definition 8. The length of a formula ψ denoted by $|\psi|$ is the number of variables and operators in ψ, i.e.:

$|\psi| = 1$ if ψ is an atomic formula

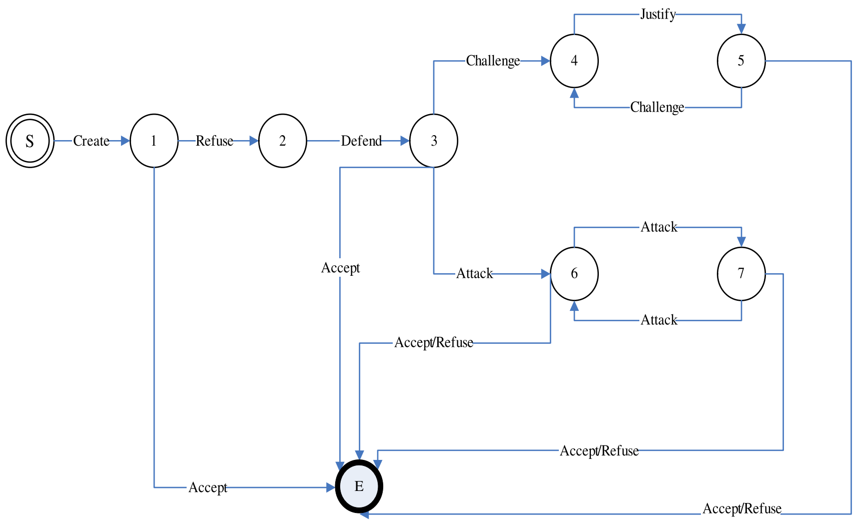


Fig. 11. The *PNAWS* protocol.

$|\neg\psi| = 1 + |\psi|$
$|\psi_1 \wedge \psi_2| = 1 + |\psi_1| + |\psi_2|$
$|\psi_1 \vee \psi_2| = 1 + |\psi_1| + |\psi_2|$
$|\therefore\psi| = 1 + |\psi|$
$|\psi_1 \therefore \psi_2| = 1 + |\psi_1| + |X^{+}(\neg\psi_1 \vee \psi_2)|$
$|X\psi| = 1 + |\psi|$ where $X \in \{X^{+}, X^{-}\}$
$|\psi_1 U \psi_2| = 1 + |\psi_1| + |\psi_2|$ where $(U,X) \in \{(U^{+},X^{+}), (U^{-},X^{-})\}$
$|PC(Ag_1,Ag_2,t,\psi)| = 1 + |\psi|$
$|C(Ag_1,SC(Ag_1,Ag_2,t,\psi))| = 1 + |PC(Ag_1,Ag_2,t,\psi)|$
$|Withdraw(Ag_1,PC(Ag_1,Ag_2,t,\psi))| = 1 + |\neg PC(Ag_1,Ag_2,t,\psi)|$
$|Satisfy(Ag_1,PC(Ag_1,Ag_2,t,\psi))| = 1 + |\psi|$
$|Violate(Ag_1,PC(Ag_1,Ag_2,t,\psi))| = 1 + |\neg\psi|$
$|Reactivate(Ag_1,PC(Ag_1,Ag_2,t,\psi))| = 1 + |PC(Ag_1,Ag_2,t,\psi)|$
$|Challenge(Ag_2,PC(Ag_1,Ag_2,t,\psi))| = 1 + |PC(Ag_2,Ag_1,t',?\psi)|$
$|Accept(Ag_2,SC(Ag_1,Ag_2,t,\psi))| = 1 + |PC(Ag_2,Ag_1,t',\psi)|$
$|Refuse(Ag_2,SC(Ag_1,Ag_2,t,\psi))| = 1 + |PC(Ag_2,Ag_1,t',\neg\psi)|$
$|Justify(Ag_1,PC(Ag_1,Ag_2,t,\phi'\therefore\psi))| = 1 + |PC(Ag_1,Ag_2,t',\phi'\therefore\psi)|$
$|Attack(Ag_2,PC(Ag_1,Ag_2,t,\phi'\therefore\psi))| = 1 + |PC(Ag_2,Ag_1,t',\phi'\therefore\neg\psi)|$
$|Defend(Ag_1,PC(Ag_1,Ag_2,t,\phi'\therefore\psi))| = 1 + |PC(Ag_1,Ag_2,t',\phi'\therefore\psi)|$

The ordering relation $\prec$ is defined as follows:

Definition 9. Let $\sigma_1 = E(\psi_1)$ and $\sigma_2 = E(\psi_2)$ be two CTL^{*CA} formulae. Then, $\sigma_1 \prec \sigma_2$ holds if

1. $\sigma_1 \leqslant \sigma_2$
2. $\sigma_1 \nleqslant \sigma_2$ and $|\psi_1| > |\psi_2|$.

where $\sigma_1 \leqslant \sigma_2$ iff $X\psi_1$ appears in ψ_2.

The first clause is used when we have a recursive formula (this means that an *until* formula). $\prec$ is irreflexive, asymmetric and transitive. The proof is straightforward from the definition since $\rangle$ and are strict ordering relations.

In what follows, the notation $\sigma_1 \rightarrow_R \sigma_2$ means that σ_1 is the parent of σ_2 using a tableau rule R. We have the following lemma (see the proof in the appendix).

Lemma 1. *Let $\sigma_1 = E(\psi_1)$ and $\sigma_2 = E(\psi_2)$ be two* DCTL$^{*}_{CAN}$ *formulae. Then:*

$\sigma_1 \rightarrow_R \sigma_2 \Rightarrow \sigma_1 \prec \sigma_2$.

To show that the ordering relation has no infinite ascending chains, we use the notion of Fischer–Ladner closure of a formula ψ ($CL(\psi)$) [15]. The idea underlying the definition of this notion is to prove that if a tableau has a root ψ, then all formulae ϕ' of this tableau have a formula in $CL(\psi)$ (i.e. $\phi' \in CL(\psi)$). Furthermore, if we prove that $CL(\psi)$ is a finite set, then we conclude that each formula appearing in a given tableau belongs to a finite set. This result will be very helpful to prove that the ordering relation $\prec$ has no infinite ascending chains.

Definition 10. Let ψ be a CTL^{*CA} formula. The Fischer–Ladner closure of ψ, $CL(\psi)$ is the smallest set such that the following hold:

If ψ is an atomic formula then $\{\psi\} \subseteq CL(\psi)$
If $\psi = \neg\psi_1$ then $CL(\psi_1) \subseteq CL(\psi)$ and $\{\neg\psi_1\} \subseteq CL(\psi)$
If $\psi = \psi_1 \wedge \psi_2$ then $CL(\psi_1) \subseteq CL(\psi)$ and $CL(\psi_2) \subseteq CL(\psi)$ and $\{\psi_1 \wedge \psi_2\} \subseteq CL(\psi)$
If $\psi = \psi_1 \vee \psi_2$ then $CL(\psi_1) \subseteq CL(\psi)$ and $CL(\psi_2) \subseteq CL(\psi)$ and $\{\psi_1 \vee \psi_2\} \subseteq CL(\psi)$
If $\psi = ?\psi_1$ then $CL(\psi_1) \subseteq CL(\psi)$ and $\{?\psi_1\} \subseteq CL(\psi)$
If $\psi = \psi_1 \therefore \psi_2$ then
$CL(\psi_1) \subseteq CL(\psi)$ and $CL(X^{+}(\neg\psi_1 \vee \psi_2)) \subseteq CL(\psi)$ and $\{\psi_1\psi_2\} \subseteq CL(\psi)$
If $\psi = X\psi_1$ then $CL(\psi_1) \subseteq CL(\psi)$ and $\{X\psi_1\} \subseteq CL(\psi)$ where $X \in \{X^{+},X^{-}\}$
If $\psi = \psi_1 U \psi_2$ then
$CL(\psi_1) \subseteq CL(\psi)$ and $CL(\psi_2) \subseteq CL(\psi)$ and $CL(X(\psi_1 U \psi_2)) \subseteq CL(\psi)$
and $\{\psi_1 U \psi_2\} \subseteq CL(\psi)$ where $(U,X) \in \{(U^{+},X^{+}), (U^{-},X^{-})\}$
If $\psi = PC(Ag_1,Ag_2,t,\psi_1)$ then
$CL(\psi_1) \subseteq CL(\psi)$ and $\{PC(Ag_1,Ag_2,t,\psi_1)\} \subseteq CL(\psi)$
If $\psi = C(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))$ then
$CL(PC(Ag_1,Ag_2,t,\psi_1)) \subseteq CL(\psi)$ and $\{C(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))\} \subseteq CL(\psi)$
If $\psi = Withdraw(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))$ then
$CL(\neg PC(Ag_1,Ag_2,t,\psi_1)) \subseteq CL(\psi)$ and
$\{Withdraw(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))\} \subseteq CL(\psi)$
If $\psi = Satisfy(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))$ then
$CL(\psi_1) \subseteq CL(\psi)$ and $\{Satisfy(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))\} \subseteq CL(\psi)$
If $\psi = Violate(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))$ then
$CL(\neg\psi_1) \subseteq CL(\psi)$ and $\{Violate(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))\} \subseteq CL(\psi)$
If $\psi = Reactivate(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))$ then
$CL(PC(Ag_1,Ag_2,t,\psi_1)) \subseteq CL(\psi)$ and
$\{Reactivate(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))\} \subseteq CL(\psi)$
If $\psi = Challenge(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))$ then

$CL(PC(Ag_1,Ag_2,t',?\psi_1)) \subseteq CL(\psi)$
and $\{Challenge(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))\} \subseteq CL(\psi)$
If $\psi = Accept(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))$ then
$CL(PC(Ag_1,Ag_2,t',\psi_1)) \subseteq CL(\psi)$
and $\{Accept(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))\} \subseteq CL(\psi)$
If $\psi = Refuse(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))$ then
$CL(PC(Ag_1,Ag_2,t',\neg\psi_1)) \subseteq CL(\psi)$
and $\{Refuse(Ag_1,PC(Ag_1,Ag_2,t,\psi_1))\} \subseteq CL(\psi)$
If $\psi = Justify(Ag_1,PC(Ag_1,Ag_2,t,\psi_2 \therefore \psi_1))$ then
$CL(PC(Ag_1,Ag_2,t',\psi_2 \therefore \psi_1)) \subseteq CL(\psi)$
and $\{Justify(Ag_1,PC(Ag_1,Ag_2,t,\psi_2 \therefore \psi_1))\} \subseteq CL(\psi)$
If $\psi = Attack(Ag_2,PC(Ag_1,Ag_2,t,\psi_2 \therefore \neg\psi_1))$ then
$CL(PC(Ag_2,Ag_1,t',\psi_2 \therefore \neg\psi_1)) \subseteq CL(\psi)$
and $\{Attack(Ag_2,PC(Ag_1,Ag_2,t,\psi_2 \therefore \neg\psi_1))\} \subseteq CL(\psi)$
If $\psi = Defend(Ag_1,PC(Ag_1,Ag_2,t,\psi_2 \therefore \psi_1))$ then
$CL(PC(Ag_1,Ag_2,t',\psi_2 \therefore \psi_1)) \subseteq CL(\psi)$
and $\{Defend(Ag_1,PC(Ag_1,Ag_2,t,\psi_2 \therefore \psi_1))\} \subseteq CL(\psi)$

Lemma 2. *Let ψ be a formula, then $CL(\psi)$ is finite and bounded in size by $2\ |\psi|$.*

The proof of this lemma is developed in the appendix. The next lemma establishes the link between tableau rules and Fischer–Ladner closure of formulae (see the proof in the appendix).

Lemma 3. *Let $\sigma_1 = E(\Phi,\psi_1)$ and $\sigma_2 = E(\Phi,\psi_2)$ be two CTL^{*CA} formulae. Then:*

$$\sigma_1 \rightarrow_R \sigma_2 \Rightarrow CL(\psi_2) \subseteq CL(\psi_1).$$

Intuitively, $\sigma_i \prec \sigma_j$ holds if σ_i is an ancestor of σ_j in some tableau, i.e. if there are rules $Ri,\ldots,Rj$ such that: $\sigma_i \rightarrow_{Ri} \sigma_{i+1} \ldots \rightarrow_{Rj} \sigma_j$. We have the following lemma (see the proof in the appendix).

Lemma 4. *The ordering relation $\prec$ has no infinite ascending chains.*

Now, we can easily prove the finiteness theorem as shown in the appendix.

Theorem 3. *For any CTL^{*CA} formula σ_1, there is a maximum height tableau has σ_1 as a root.*

Let us now discuss the worst-case time complexity of our model checking (see the proofs in the appendix).

Lemma 5. *Let ψ be a CTL^{*CA} formula, and let $B_\psi = \langle Q, l, \rightarrow, q_0, F\rangle$ be the ABTA obtained by the translation procedure. Then $|B_\psi| < 2^{|\psi|}$.*

The complexity of the transition procedure is thus exponential in the size of the formula ($O(2^{|\psi|})$). However, if ψ is a CTL^{CA} formula, $|B_\psi|$ is bounded by $|\psi|$. The complexity is then linear in the size of the formula. This result follows from the fact that in CTL^{CA} we have only state formulae.

Lemma 6. *Let $T = \langle S, Lab, \wp, L, Act, \rightarrow, s_0\rangle$ be a transition system for a dialogue game, and let $B_\psi = \langle Q, l, \rightarrow, q_0, F\rangle$ be an ABTA for ψ. The time complexity of the model checking algorithm is bounded by $|T| \times |B_\psi|$ where $|T| = |S| + |\wp| + |\rightarrow|$ and $|\wp|$ is the number of sub-states in all structure transition systems of T.*

The worst-case time complexity of our model checking technique is therefore linear in the size of the model and exponential in the size of the formula to be checked.

8. Case study

8.1. Protocol description and verification

In this section, we will show how our model checking technique works by using it to verify a typical and concrete agent-based protocol: the *PNAWS* protocol (persuasion/negotiation for agent-based web services) [4]. We also show the feasibility of this technique by discussing its implementation details using a modified and enhanced Concurrency Workbench of New Century (CWB-NC) [12,27]. In fact, the feasibility of the approach is guaranteed by the feasibility of CWB-NC, which is used to check many large-scale protocols in networking communication and process control systems. *PNAWS* is a dialogue game-based protocol allowing Web services implemented as argumentative agents to interact with each other in a negotiation setting. Agents can negotiate their participation in composite Web services and persuade each other to perform some actions. *PNAWS* is specified using two special moves: *refusal* and *acceptance* as well as five dialogue games: *entry game* (to open the interaction), *defense game*, *challenge game*, *justification game*, and *attack game*. *PNAWS* can be defined using a BNF-like grammar where "|" is the choice symbol and ";" the sequence symbol as follows:

$$\begin{aligned}
&PNAWS = \text{entry game; defense game;} WSDG\\
&WSDG = \text{acceptance move} \mid Ch \mid Att\\
&Ch = \text{challenge game; justification game;}(WSDG \mid \text{refusal move})\\
&Att = \text{attack game;}(WSDG \mid \text{refusal move})
\end{aligned}$$

Each game is specified by a set of moves using a set of logical rules. For space limit reasons, we will not describe these rules. However, we provide a graphical representation, which is enough to understand the protocol dynamics. Fig. 11 illustrates this representation as a finite state machine. Many properties can be checked in this protocol, such as deadlock freedom (a safety property), and liveness (something good will eventually happen). Deadlock freedom can be expressed in our CTL^{*CA} logic as follows:

$$AG^+(Act(Ag_i, PC(Ag_1, Ag_2, \mathscr{P}))$$

which states that there is always a possibility for an action. An example of liveness can be expressed by the following formula:

$$EF^+(Accept(Ag_2, PC(Ag_1, Ag_2, ?)) \vee Refuse(Ag_2, PC(Ag_1, Ag_2, ?)))$$

which states that there is a possibility to achieve some good states (accept or refuse). Another example of liveness property is given by the following formula stating that if there is a challenge, a justification will eventually follow:

$$\begin{aligned}
&AG^+(Challenge(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \Rightarrow\\
&F^+Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi'\phi)))
\end{aligned}$$

The two first properties are relatively easy to check. We focus in this section on the third property. In order to simplify the formula, we use *Ch* for *Challenge* and *Jus* for *Justify*. The first step in our technique is the transformation of the formula to a tableau according to the translation procedure presented in Section 6.2. The tableau of this formula is illustrated by Fig. 12. The second step is building the associated ABTA using the same translation procedure. The ABTA of this formula is given by Fig. 13. This formula is equivalent to:

$$AG^+(\neg Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \vee F^+Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))$$

To build the tableau, the first rule we can apply is $R6$ labeled by "$\neg$". We obtain then the formula (2) of Fig. 12. From this formula we obtain the formula Φ that we consider in order to compute the acceptance states:

$$\begin{aligned}
\Phi = &F^+(Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi))\\
&\wedge G^+(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi'\phi))))
\end{aligned}$$

$$\neg : AG^{+}(\neg Ch(Ag_2,PC(Ag_1,Ag_2,t,\phi)) \vee F^{+}Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi)))\ (1)$$

$$\vee : EF^{+}(Ch(Ag_2,PC(Ag_1,Ag_2,t,\phi)) \wedge G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi'))))\ (2)$$

$$<Ch>: E(Ch(Ag_2,PC(Ag_1,Ag_2,t,\phi)) \wedge G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (3) \qquad <X^{+}>: EX^{+}(F^{+}(Ch(Ag_2,PC(Ag_1,Ag_2,t,\phi)) \wedge G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi)))))\ (4)$$

$$[PC_{Ag2}]: E(PC(Ag_2,Ag_1,t,?\phi) \wedge G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi'))))\ (5) \qquad EF^{+}(Ch(Ag_2,PC(Ag_1,Ag_2,t,\phi)) \wedge G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (2)$$

$$?: E((?\phi) \wedge G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (6)$$

$$<\equiv>: E(\phi \wedge G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (7)$$

$$\varphi\ (8) \qquad \vee : E(G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (9)$$

$$<\neg Jus>: E(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi)), X^{+}G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (10)$$

$$[PC_{Ag1}]: E(PC(Ag_1,Ag_2,t,\phi' \therefore \phi), X^{+}G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (11)$$

$$\wedge : E(\phi' \therefore \phi, X^{+}G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (12)$$

$$<\equiv>: E(\phi', X^{+}(\neg\phi' \vee \phi), X^{+}G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (13)$$

$$\phi'\ (14) \qquad X^{+}: E(X^{+}(\neg\phi' \vee \phi), X^{+}G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (15)$$

$$<\equiv>: E((\neg\phi' \vee \phi), X^{+}G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (16)$$

$$\neg\varphi' \vee \varphi\ (17) \qquad X^{+}: E(X^{+}G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (18)$$

$$E(G^{+}(\neg Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi' \therefore \phi))))\ (9)$$

Fig. 12. The tableau for $AG^{+}(Ch(Ag_2,PC(Ag_1,Ag_2,t,\ \phi)) \Rightarrow +Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi'\phi)))$.

In the ABTA of Fig. 13, state (1) and states from (3) to (18) are the acceptance states according to Definition 7. States (2) and (4) are not acceptance states. Because only the first state is labeled by $\neg$, all finite and infinite paths are negative paths. Consequently, the only infinite path that is a valid proof of the formula Φ is $(1,(2,4)^{*})$. In this path there is no acceptance state that occurs infinitely often. Therefore, this path satisfies the Büchi condition. The path visiting the state (3) and infinitely often the state (9) does not satisfy the formula because there is a challenge action (state (3)), and globally no justification action of the content of the challenged propositional commitment (state (9)).

Fig. 14 illustrates the automaton $B_{\otimes}$ resulting from the product of the transition system of Fig. 11 to which we add the internal states to describe the syntax exactly as illustrated in Fig. 5, Section 5.1 and the ABTA of Fig. 13. We will use TS [9] to denote the protocol and ABTA [11] to denote the ABTA of Fig. 13. In order to check if the language of this automaton is empty, we check if there is a successful run. The idea is to verify if $B_{\otimes}$ contains an infinite path visiting the state (3) and infinitely often the state (9) of ABTA [11]. If such a path exists, then we conclude that the formula is not satisfied by TS [9]. Indeed, the only infinite path of $B_{\otimes}$ is successful because it does not touch any accepted state and all leaves are also successful. For instance, the leaf labeled by $(\langle Ch\rangle,s_0)$ is successful since there is no state s_i such that $s_0 \rightarrow {}^{Ch}s_i$. The leaf labeled by $(\neg\phi' \vee \phi,s_{3,4})$ is successful because it is a positive leaf and $s_{3,4} \vDash \neg\phi\ ' \vee \phi$. Therefore, TS[9] is accepted by ABTA[11]. Consequently, TS[9] satisfies the formula and respects the structure of challenge and justification actions.

8.2. Implementation of the case study

To show the feasibility of our technique, we implemented this case study using a modified and enhanced version of the Concurrency Workbench of New Century (CWB-NC) [12,27]. This model checker supports GCTL*, which is close to our logic (without propositional commitments) and allows modeling concurrent systems using process algebra calculus of communicating systems (CCS) developed by Milner [23]. CCS language is a paradigmatic process algebras language, which is a prototype specification language for reactive systems. For this reason, CCS language can be used not only to describe implementations of processes, but also specifications of their expected behaviors.

To use CCS as the design language to descript *PNAWS* protocol, we need first to describe CCS formally for this protocol. Let *A*be the set of actions performed on propositional commitments we consider in our logic. With every $a \in A$ we associate a complementary action $'a$. Intuitively, an action *a*will represent the receipt of an input action, while $'a$ will represent the deposit of an output action. The syntax of CCS for *PNAWS* protocol is given by the following BNF grammar:

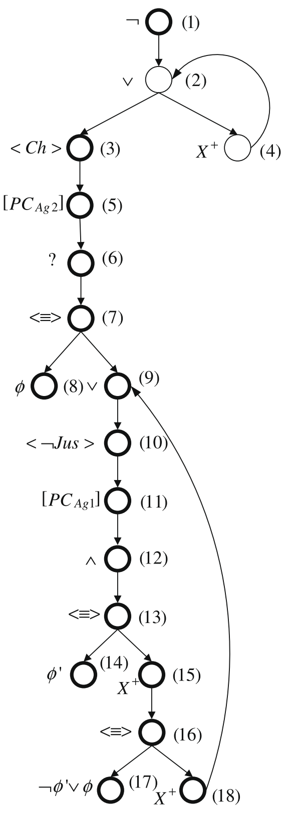


Fig. 13. The ABTA for The formula $AG^{+}(Ch(Ag_2,PC(Ag_1,Ag_2,t,\phi))F^{+}Jus(Ag_1,PC(Ag_1,Ag_2,t,\phi',\therefore,\phi)))$ using the translation procedure (Section 6.2).

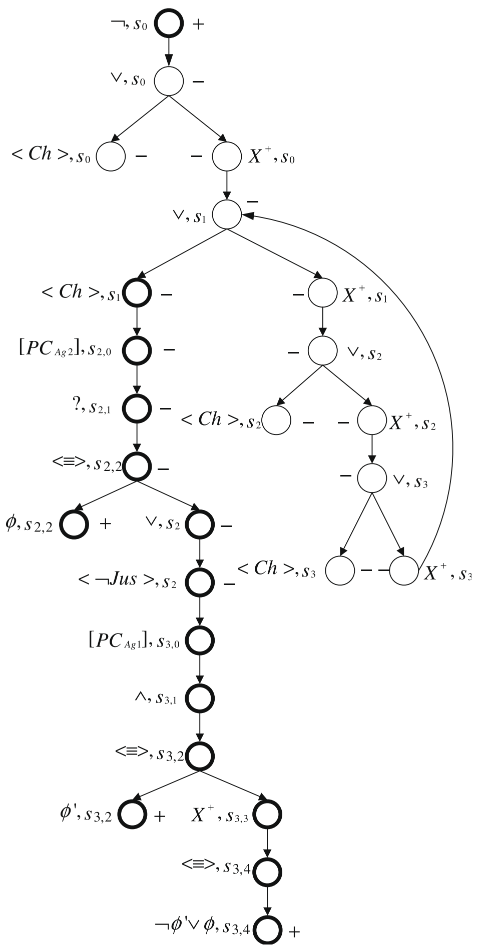


Fig. 14. The ABT product graph.

$$P ::= nil \mid \alpha(\phi).P \mid (P+P) \mid (P \mid P) \mid proc\, C = P$$

"." represents the prefixing operator, "+" is the choice operator, "|" is the parallel operator and "*proc* = " is used for defining processes. The semantics can be defined using operational semantics in the usual way. $\alpha(\phi).P$ is the processes of performing the action α on the propositional commitment content ϕ and then evolves into process P. For simplification reasons, we consider only the commitment content and we omit the other arguments. In addition, we abstract away from the internal states used to check the content syntax, and we focus only on verifying the properties from a semantic perspective. $P+Q$ is the process which non-deterministically makes the choice of evolving into either P or Q.$P|Q$ is the process which evolves in parallel into P and Q.

To implement *PNAWS* we need to model the protocol and the agents using this protocol. For this reason, we need to define two processes: the *states process* describing the protocol dynamics and the *agent process* describing the agent legal decisions. These two processes are as follows:

The definition of the states process:

```
proc Spec=create(φ).S1
proc Accept=accept(φ).Spec
proc Accept′=′accept(φ).Spec
proc Refuse=refuse(φ).Spec
proc Refuse′=′refuse(φ).Spec
proc S1=′refuse(φ).S2 + Accept′
proc S2=defend(φ′).S3
proc S3=′challenge(φ′).S4 +′attack(φ).S6 +′accept(φ
′).Spec
proc S4=justify(φ).S5
proc S5=′challenge(φ).S4 +′Accept +′Refuse
proc S6=attack(φ′).S7 + Accept +Refuse
proc S7=′attack(φ).S6 +′accept(φ′).Spec +’refuse(φ
′).Spec
set Internals={create, challenge, justify, accept,
refuse, attack, defend}
```

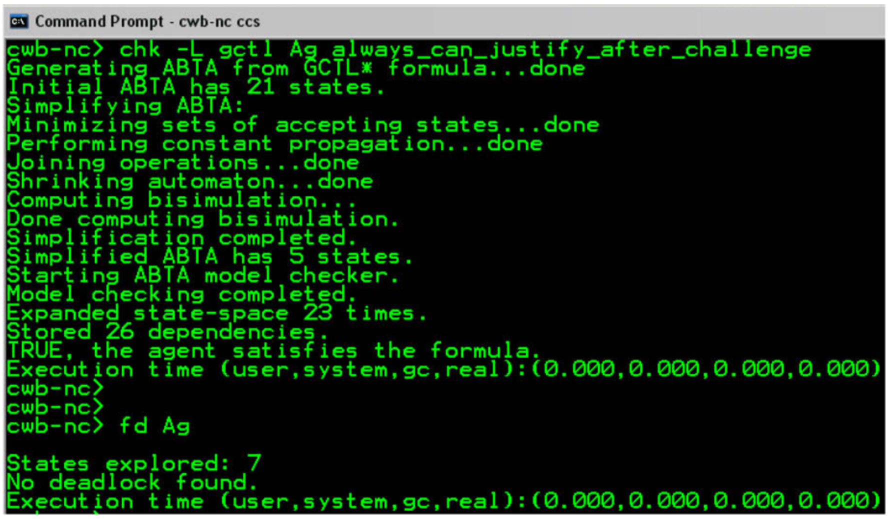
The definition of the agent process:

```
proc Ag=′create(φ).Ag +
  create(φ).(′refuse(φ).Ag +′accept(φ).Ag) +
  refuse(φ).(Ag +′defend(φ′).Ag) +
  defend(φ′).(′challenge(φ′).Ag   +   ′attack(φ).Ag
  +′accept(φ′).Ag) +
  challenge(φ).′justify(φ′).Ag +
  justify(φ′).(′challenge(φ′).Ag   +   ′accept(φ′).Ag
  +′refuse(φ′).Ag) +
  attack(φ′).(′attack(φ).Ag     +     ′accept(φ′).Ag
  +′refuse(φ′).Ag) +
  accept(φ).Ag
```

Fig. 15 illustrates the result of checking the liveness property discussed in the previous section stating that if there is a challenge, a justification will eventually follow. The figure illustrates also the result of checking deadlock property. The other properties (reachability of accept and refuse) are also successfully checked.

9. Conclusion and future work

In this paper, we have addressed the verification problem of communicating agent-based systems, in which knowledge-driven



```
Command Prompt - cwb-nc ccs
cwb-nc> chk -L gctl Ag always_can_justify_after_challenge
Generating ABTA from GCTL* formula...done
Initial ABTA has 21 states.
Simplifying ABTA:
Minimizing sets of accepting states...done
Performing constant propagation...done
Joining operations...done
Shrinking automaton...done
Computing bisimulation...
Done computing bisimulation.
Simplification completed.
Simplified ABTA has 5 states.
Starting ABTA model checker.
Model checking completed.
Expanded state-space 23 times.
Stored 26 dependencies.
TRUE, the agent satisfies the formula.
Execution time (user,system,gc,real):(0.000,0.000,0.000,0.000)
cwb-nc>
cwb-nc>
cwb-nc> fd Ag

States explored: 7
No deadlock found.
Execution time (user,system,gc,real):(0.000,0.000,0.000,0.000)
```

Fig. 15. Result of checking liveness and deadlock properties.

agents communicate by reasoning about dialogue game protocols. The proposed protocol assumes only two agents communicating in a synchronous way. We proposed a new model checking technique allowing for the verification of both the correctness of the protocols and the agents' compliance to the structure of the communicative acts. This technique uses a combination of an automata-based and a tableau-driven algorithm to verify temporal and action specification. The formal properties to be verified are expressed in CTL^{*CA} logic and translated to ABTA using tableau rules. We provided correctness and completeness results and we proved that this model checking algorithm working on a product graph is an efficient on-the-fly procedure that always terminates. To show the soundness and feasibility of the proposed technique, we applied it to a concrete case study from the literature describing the verification of some properties in an agent-based negotiation protocol.

For future work, we intend to consider concurrency and multi-party issues for agent communication from design and verification points of view. Designing and verifying concurrent dialogue systems for multi-agents raise many problems such as synchronization and fairness. Applying the on-the-fly tableau technique proposed in this paper could be a good choice for small and medium systems. However, because the number of possible executions in a concurrent setting could be very large, applying symbolic model checking seems to be promising since this technique is suitable when explosion state becomes frequent.

Acknowledgements

We would like to thank the editors and anonymous reviewers for their valuable comments and helpful suggestions for improvements. We also thank the Natural Sciences and Engineering Research Council of Canada (NSERC), le Fond Québécois de la Recherche sur la Nature et les Technologies (NATEQ) and Concordia University (seed funding) for their financial support.

Appendix

Proof of Lemma 1. The proof is based on the analysis of the different cases of our tableau rules. Most cases are straightforward. Here we only consider rules $R7$, $R25$, and $R27$.

$R = R7$:

$\sigma_1 \to_R \sigma_2$

$\Rightarrow \sigma_1 = E(\Phi, C(Ag_1, PC(Ag_1, Ag_2, t, \psi), \sigma_2 = E(\Phi, SC(Ag_1, Ag_2, t, \psi))$

$\Rightarrow \sigma_1 \prec \sigma_2$ (from the definition of $\prec$ and the fact that

$| C(PC(Ag_1, Ag_2, t, \psi) |= 1+ | PC(Ag_1, Ag_2, t, \psi) |$

$>| PC(Ag_1, Ag_2, t, \psi) |$)

$R = R25$:

$\sigma_1 \to_R \sigma_2$

$\Rightarrow \sigma_1 = E(\Phi, \psi_1\psi_2), \sigma_2 = E(\Phi, \psi_1, X^+(\neg\psi_1 \vee \psi_2))$

$\Rightarrow \sigma_1 \prec \sigma_2$ (from the definition of $\prec$ and the fact that

$| \psi_1\psi_2 |= 1+ | \psi_1 | + | X^+(\neg\psi_1 \vee \psi_2) |$

$R = R27$:

$\sigma_1 \to_R \sigma_2$

$\Rightarrow \sigma_1 = E(\Phi, \psi_1 U^+ \psi_2), \sigma_2 = E(\Phi, \psi_2)$ or$E(\Phi, \psi_1, X^+(\psi_1 U^+ \psi_2))$

$\Rightarrow \sigma_1 \prec \sigma_2$ (from the definition of $\prec$ and the fact that$\sigma_1 \leqslant \sigma_2$ $\square$

Proof of Lemma 2. The proof is based on the induction of the structure of ψ. Most cases are straightforward. Here we only consider the four following cases:

(1) $\psi = X\psi_1$, where $X \in \{X^+, X^-\}$.We have:

$CL(X\psi_1) = \{X\psi_1\} \cup CL(\psi_1)$

Therefore:

$| CL(X\psi_1) |= 1+ | CL(\psi_1) |$

Then, by using the induction hypothesis, we conclude that:

$| CL(X\psi_1) | \leqslant 1 + 2 | \psi_1 | \leqslant 2(1+ | \psi_1 |)$

Then, by using Definition 8 we obtain:

$| CL(X\psi_1) | \leqslant 2 | X\psi_1 |$

(2) $\psi = \psi_1 U \psi_2$, where $U \in \{U^+, U^-\}$. We have:

$$\begin{aligned} CL(\psi_1 U \psi_2) &= \{\psi_1 U \psi_2\} \cup CL(\psi_1) \cup CL(\psi_2) \cup CL(X(\psi_1 U \psi_2)) \\ &= \{\psi_1 U \psi_2\} \cup CL(\psi_1) \cup CL(\psi_2) \cup \{X(\psi_1 U \psi_2)\} \end{aligned}$$

Therefore:

$| CL(\psi_1 U \psi_2) |= 2 | CL(\psi_1) | + | CL(\psi_2) |$

Then, by using the induction hypothesis and the previous case, we conclude that:

$| CL(\psi_1 U \psi_2) | \leqslant 2 + 2 | \psi_1 | + 2 | \psi_2 | + | X(\psi_1 U \psi_2) |$

Then, by using Definition 8 we obtain:

$| CL(\psi_1 U \psi_2) | \leqslant 2 | \psi_1 U \psi_2 |$

(3) $\psi = PC(Ag_1, Ag_2, t, \psi_1)$ We have:

$CL(PC(Ag_1, Ag_2, t, \psi_1)) = \{PC(Ag_1, Ag_2, t, \psi_1)\} \cup CL(\psi_1)$

Therefore:

$| CL(PC(Ag_1, Ag_2, t, \psi_1)) | = 1 + | CL(\psi_1) |$

Then, by using the induction hypothesis, we conclude that:

$| CL(PC(Ag_1, Ag_2, t, \psi_1)) | \leqslant 1 + 2 | \psi_1 |$

Then, by using Definition 8 we obtain:

$| CL(PC(Ag_1, Ag_2, t, \psi_1)) | \leqslant 2 | PC(Ag_1, Ag_2, t, \psi_1) |$

(4) $\psi = C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))$ We have:

$$CL(Create(Ag_1, PC(Ag_1, Ag_2, t, \psi_1)))$$
$$= \{C(PC(Ag_1, Ag_2, t, \psi_1))\} \cup CL(PC(Ag_1, Ag_2, t, \psi_1))$$

Therefore:

$| CL(C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))) | = 1 + 2 | CL(PC(Ag_1, Ag_2, t, \psi_1)) |$

Then, by using the previous case, we conclude that:

$| CL(C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1)) | \leqslant 1 + 2 | PC(Ag_1, Ag_2, t, \psi_1) |$

Then, by using Definition 8 we obtain:

$| CL(C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))) | \leqslant 2 | C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))$
$|$ □

Proof of Lemma 3. The proof is based on the case analysis of the rule R. Most cases are straightforward. Here we consider the rules $R7$, $R25$, and $R27$.

$R = R7$:
$$\sigma_1 \rightarrow_R \sigma_2$$
$$\Rightarrow E(\Phi, \psi_1) = E(\Phi, C(Ag_1, PC(Ag_1, Ag_2, t, \psi),$$
$$E(\Phi, \psi_2) = E(\Phi, PC(Ag_1, Ag_2, t, \psi))$$
$$\Rightarrow (\text{Definition of } CL(C(Ag_1, PC(Ag_1, Ag_2, t, \psi)))$$
$$CL(\psi_2) \subseteq CL(\psi_1)$$

$R = R25$:
$$\sigma_1 \rightarrow_R \sigma_2$$
$$\Rightarrow E(\Phi, \psi_1) = E(\Phi, \psi \therefore \phi')), E(\Phi, \psi_2) = E(\Phi, \psi, X^+(\neg\psi \vee \phi'))$$
$$\Rightarrow CL(\psi_1) = \{\psi\phi'\} \cup CL(\psi) \cup CL(X^+(\neg\psi \vee \phi'))$$
$$\Rightarrow CL(\psi_2) \subseteq CL(\psi_1)$$

$R = R27$:
$$\sigma_1 \rightarrow_R \sigma_2$$
$$\Rightarrow E(\Phi, \psi_1) = E(\Phi, \psi U^+ \phi'), E(\Phi, \psi_2) = E(\Phi, \phi') \text{ or } E(\Phi, \psi, X^+(\psi U^+ \phi'))$$
$$\Rightarrow CL(\psi_1) = \{\psi U^+ \phi'\} \cup CL(\psi) \cup CL(\phi') \cup CL(X^+(\psi U^+ \phi'))$$
$$\Rightarrow CL(\psi_2) \subseteq CL(\psi_1) \quad \square$$

Proof of Lemma 4. Suppose that there exists an infinite chain: $\sigma_1 \prec \sigma_2 \prec \ldots$From Lemma 3, it follows that $CL(\psi_i) \subseteq CL(\psi_{i-1}) \subseteq \ldots CL(\psi_1)$ Since $CL(\psi_1)$ is finite (from Lemma 2), it follows that:

$$\exists j, \forall k \geqslant j, CL(\psi_k) = CL(\psi_j) \quad \text{with } \sigma_j \prec \sigma_{j+1} \prec \ldots \ \sigma_k \prec \ldots$$

However, this is contradictory (from Lemma 3). □

Proof of Lemma 5. Suppose that there exists a tableau with root σ_1 having an infinite path:

$$\sigma_1 \rightarrow_{Ri} \sigma_2 \rightarrow_{Rj} \sigma_3 \ldots$$

where $Ri, Rj, \ldots \in \{R1, \ldots, R27\}$. Then, from Lemma 1 and from the fact that the ordering relation $\prec$ is transitive (since $\langle$ is transitive), it follows that there exists an infinite chain:

$$\sigma_1 \prec \sigma_2 \prec \ldots$$

However this is contradictory from Lemma 4. □

Proof of Lemma 6. From the transition procedure, each formula ϕ' in the tableau is a sub-formula of ψ. The formula ψ is decomposed into a set of sub-formulae using the tableau rules. The nodes in the ABTA are labeled by the operators from the sub-formulae and there is a transition from a node φ to a node φ'' if the formula corresponding to φ'' is a sub-formula of the one corresponding to φ. Since for every sub-formula ϕ' of ψ we have $\phi' \subseteq CL(\psi)$ and $|CL(\psi)| \langle |\psi|$ (from Lemma 2), it follows that $|B_\psi| \langle 2^{|\psi|}$. □

Proof of Lemma 7. The algorithm is based on a product graph of the ABTA B_ψ and the transition system T. The size of this product is bounded by $|T| \times |B_\psi|$. Like the algorithms proposed in (Courcoubetis et al. [14]) and (Bhat and Cleaveland [9]), our algorithm marks nodes and determines if an accepting state is reachable from itself. This algorithm visits each state once and there are $|S| \times |Q|$ recursive calls to a depth-first search algorithm. We note also that the ABTA we use is an *and-restricted* one. In an and-restricted ABTA only one of the children of a node labeled by $\wedge$ can have his truth values determined by recursive calls to search algorithm (Bhat and Cleaveland [9]). The run time of the algorithm is thus proportional to the size of the product graph, i.e. $O(|T| \times |B_\psi|)$. □

References

[1] K. Adi, M. Debbabi, M. Mejri, A new logic for electronic commerce protocols, Theoretical Computer Science 291 (2003) 223–283.

[2] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, C. Schifanella, Verifying Protocol Conformance for Logic-based Communicating Agents, Declarative Agent Languages and Technologies, Lecture Notes in Computer Science, vol. 3487, Springer, Berlin, 2005. pp. 82–97.

[3] J. Bentahar, A Pragmatic and Semantic Unified Framework for Agent Communication, Ph.D. Thesis, Department of Computer Science and Software Engineering, Laval University, Canada, 2005.

[4] J. Bentahar, Z. Maamar, D. Benslimane, P. Thiran, An argumentation framework for communities of web services, IEEE Intelligent Systems 22 (6) (2007) 75–83.

[5] J. Bentahar, B. Moulin, J.-J.Ch. Meyer, A Tableau Method for Verifying Dialogue Game Protocols for Agent Communication, Declarative Agent Languages and Technologies, Lecture Notes in Computer Science, vol. 3904, Springer, Berlin, 2006. pp. 223–244.

[6] J. Bentahar, B. Moulin, J.-J.Ch. Meyer, B. Chaib-draa, A Computational Model for Conversation Policies for Agent Communication, Computational Logic in Multi-agent Systems, Lecture Notes in Computer Science, vol. 3487, Springer, Berlin, 2005. pp. 178–195.

[7] O. Bernholtz, M.Y. Vardi, P. Wolper, An automata-theoretic approach to branching-time model checking, Computer aided verification, in: D.L. Dill (Ed.), Lecture Notes in Computer Science, vol. 818, Springer, Berlin, 1994, pp. 142–155.

[8] G. Bhat, R. Cleaveland, Efficient model checking via the equational µ-calculus, in: The 11th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press, 1996, pp. 304–312.

[9] G. Bhat, R. Cleaveland, A. Groce, Efficient model checking via Büchi tableau automata, Computer-aided verification, in: G. Berry, H. Comon, A. Finkel (Eds.), Lecture Notes in Computer Science, vol. 2102, Springer, Berlin, 2001, pp. 38–52.

[10] R.H. Bordini, M. Fisher, C. Pardavila, M. Wooldridge, Model checking AgentSpeak, in: Proceedings of the 2nd International Joint Conference On Autonomous Agents and Multi Agent Systems, 2003, pp. 409–416.

[11] R.H. Bordini, W. Visser, M. Fisher, C. Pardavila, M. Wooldridge, Model checking multi-agent programs with CASP, Computer-aided verification, in: W.A. Hunt, F. Somenzi (Eds.), Lecture Notes in Computer Science, vol. 2725, Springer, Berlin, 2003, pp. 110–113.

[12] R. Cleaveland, S. Sims, Generic tools for verifying concurrent systems, Science of Computer Programming 41 (1) (2002) 39–47.
[13] R. Cleaveland, Tableau-based model checking in the propositional mu-calculus, Acta Informatica 27 (8) (1990) 725–747.
[14] C. Courcoubetis, M.Y. Vardi, P. Wolper, M. Yannakakis, Memory efficient algorithms for verification of temporal properties, Formal Methods in System Design 1 (2–3) (1992) 275–288.
[15] E.A. Emerson, C. Jutla, A.P. Sistla, On Model-checking for Fragments of μ-Calculus, Computer Aided Verification, Lecture Notes in Computer Science, vol. 697, Springer, Berlin, 1993, pp. 385–396.
[16] U. Endriss, N. Maudet, F. Sadri, F. Toni, Protocol conformance for logic-based agents, in: Proceedings of the 18th International Joint Conference on Artificial Intelligence, 2003, pp. 679–684.
[17] H. Fujita, Special issue on "techniques to produce Intelligent_Secure software, Knowledge-Based Systems 20(7) (2007) 614–616.
[18] L. Giordano, A. Martelli, C. Schwind, Verifying Communicating Agents by Model Checking in a Temporal Action Logic, Logics in Artificial Intelligence (JELIA'04), Lecture Notes in Computer Science, vol. 3229, Springer, Berlin, 2004, pp. 57–69.
[19] M.-P. Huget, M. Wooldridge, Model checking for ACL compliance verification, Advances in agent communication, in: F. Dignum (Ed.), Lecture Notes in Computer Science, vol. 2922, Springer, Berlin, 2004, pp. 75–90.
[20] M. Kacprzak, A. Lomuscio, W. Penczek, Verification of multiagent systems via unbounded model checking, in: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems, 2004, pp. 638–645.
[21] A. Lomuscio, C. Pecheur, F. Raimondi, Automatic verification of knowledge and time with NuSMV, in: International Joint Conference on Artificial Intelligence, 2007, pp. 1384–1389.
[22] P. McBurney, S. Parsons, Games that agents play: a formal framework for dialogues between autonomous agents, Journal of Logic, Language and Information 11 (3) (2002) 315–334.
[23] R. Milner, Operational and algebraic semantics of concurrent processes, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier, Amsterdam, 1990, pp. 1201–1242.
[24] F. Raimondi, A. Lomuscio, Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams, Journal of Applied Logic 5 (2) (2007) 235–251.
[25] F. Sadri, F. Toni, P. Torroni, Logic agents, dialogues and negotiation: an abductive approach, in: Proceedings of the Symposium on Information Agents for E-Commerce, Artificial Intelligence and the Simulation of Behaviour Conference, 2001.
[26] M. Vardi, P. Wolper, An automata-theoretic approach to automatic program verification, in: Symposium on Logic in Computer Science, 1986, pp. 332–344.
[27] D. Zhang, R. Cleaveland, E.W. Stark, The Integrated CWB-NC/PIOATool for Functional Verification and Performance Analysis of Concurrent Systems, Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 2619, Springer, Berlin, 2003, pp. 431–436.