A Tableau Method for Verifying Dialogue Game Protocols for Agent Communication

Jamal Bentahar[1], Bernard Moulin[1], and John-Jules Ch. Meyer[2]

[1] Laval University, Department of Computer Science and Software Engineering, Sainte-Foy, QC, G1K 7P4, Canada
{jamal.bentahar,bernard.moulin}@ift.ulaval.ca

[2] University Utrecht, Department of Computer Science, The Netherlands
jj@cs.uu.nl

Abstract. In this paper, we propose a new tableau-based model checking technique for verifying dialogue game protocols for agent communication. These protocols are defined using our social commitment-based framework for agent communication called Commitment and Argument Network (CAN). We use a variant of CTL* (ACTL*) for specifying these protocols and the properties to be verified. This logic extends CTL* by allowing formulae to constrain actions as well as states. The verification method is based on the translation of formulae into a variant of alternating tree automata called Alternating Büchi Tableau Automata (ABTA). We propose a set of tableau rules (inference rules) for specifying this translation procedure. Unlike the model checking algorithms proposed in the literature, the algorithm that we propose in this paper allows us not only to verify if the dialogue game protocol (the model) satisfies a given property, but also if this protocol respects the tableau rules-based decomposition of the action formulae. This algorithm is an on-the-fly efficient algorithm.

1 Introduction

The success of Multi-Agent Systems (MAS) is primarily based on the ability of agents to communicate. Research in this domain has received much attention during the past years. Particularly, several dialogue game protocols have been proposed for specifying agent communication interactions [5, 21, 22, 27]. These games aim at offering more flexibility by combining different small games to construct complete and more complex protocols. Dialogue games can be thought of as interaction games in which each agent plays a move in turn by performing utterances according to a pre-defined set of rules.

From another point of view, formal verification methods became usable by industry quite recently and there is a growing demand for professionals able to apply them. In this domain, model-based techniques are verification approaches of system properties. In these approaches, the system is represented by a finite model M using an appropriate logic. The specification is again represented by a formula ϕ and the verification method consists of computing whether the model M satisfies ϕ or not.

Recently, the verification of MAS has become an attractive field of research [3, 9, 10, 19, 20, 24, 26, 25, 30, 33]. However, in the domain of agent communication, only few research work tried to address the verification of agent communication protocols [1, 2, 15, 17, 18, 32].

In this paper, we propose a model checking-based verification of dialogue game protocols for agent communication using an action and temporal logic (ACTL*). Using a model checking technique for this verification is motivated by the fact that model-checking is a successful technique for automatically and computationally verifying protocol specifications using a suitable logic. This technique can be used to verify the protocol correctness in the sense that the protocol satisfies the expected properties. In our work, this technique allows us to verify agent communication properties specified using our ACTL* logic. Therefore, we can specify the protocol in a logical way and verify its correctness in terms of the satisfaction of the expected properties. The definition of a new logic is motivated by the fact that our dialogue game protocols should be specified using not only temporal properties, but also action properties. In addition, in these protocols, actions that agents perform by communicating are expressed in terms of "Social Commitments" (SC) and arguments. These protocols are specified as transition systems (TS) using ACTL* logic and our Commitment and Argument Network (CAN) [6]. These TS are labeled with actions that agents apply to SC and to SC contents [13, 16, 28].

The model checking technique that we propose is based on the translation of the formula expressing the property to be verified into a variant of alternating tree automata called Alternating Büchi Tableau Automata (ABTA). This technique is an extension of the ABTA-based algorithm for CTL* proposed in [7]. The choice of this technique is motivated by the fact that unlike other model checking techniques, this technique allows us to check temporal and action formulas. In addition, this technique is one of the most efficient techniques proposed in the literature. The translation procedure uses a set of inference rules called tableau rules. Like automata-based model checking of PLTL, our technique is based on the product graph of the model and the automata representing the formula to be verified (Fig. 1). This technique allows us to verify not only that the dialogue game protocol satisfies a given property, but also that this protocol respects the decomposition rules of the communicative acts. Consequently, if agents respect these protocols, then they also respect the decomposition semantics of the communicative acts. Thus, we have only one procedure to verify:

1. the correctness of the protocols relative to the properties that the protocols should satisfy;
2. the conformance of agents to the decomposition semantics of the communicative acts.

To our knowledge, until now there is no work that addressed the verification problem of dialogue game protocols. Indeed, the contributions of this paper are: (1) a specification of these protocols using TS and the CAN framework; (2) an automata and tableau-based technique to check if a dialogue game protocol

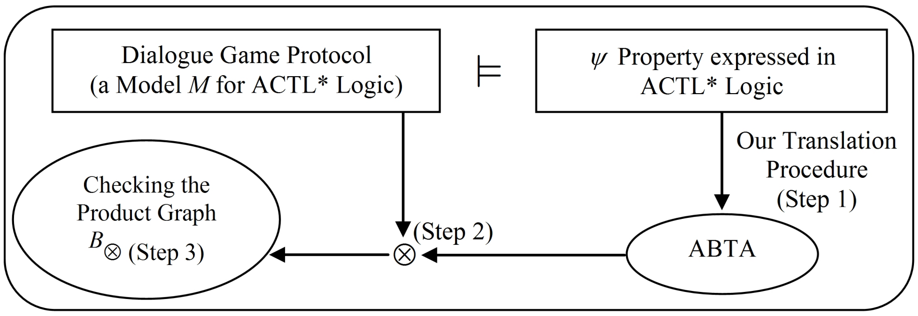
Dialogue Game Protocol
(a Model M for ACTL* Logic)
$\models$
ψ Property expressed in ACTL* Logic
Our Translation Procedure (Step 1)
Checking the Product Graph $B_\otimes$ (Step 3)
(Step 2)
$\otimes$
ABTA

Fig. 1. Our model checking approach

satisfies the specifications and their semantics. These two verifications are done at the same time.

The rest of this paper is organized as follows. Section 2 introduces tableau-based algorithms for model checking, which we use in our verification procedure. Section 3 presents our ACTL* logic. In Section 4, we use this logic to define the TS that we use to specify dialogue game protocols. The problem of verifying these protocols is addressed in Section 5. The tableau rules associated to ACTL* logic, the ABTA's definition that we use in our verification technique, and some running examples of the model checking steps are presented in this section. Section 6 presents related work and concludes the paper by identifying some directions for future work.

2 Tableau-based Algorithms for Model Checking

Unlike traditional proof systems which are bottom-up approaches, tableau-based algorithms used for model checking work in a *top-down* or *goal-oriented* fashion [11]. In the tableau-based approach, *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 (a formula), and we apply a tableau rule and determine the sub-goals (sub-formulae) to be proven. The tableau 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 to be checked is explored in a need-driven fashion [7]. The model checking algorithm searches only the part of the state space that needs to be explored to prove or disprove a certain formula. The state space is constructed while the algorithm runs. This kind of model checking algorithms is referred to as *on-the-fly* or *local* algorithms [7, 8, 11, 29].

The tableau decision algorithm that we use in our verification technique provides a systematic search for a model which satisfies a particular formula expressed using ACTL* logic. It is a graph construction algorithm. Nodes of the graph are sets of ACTL* formulae and tableau rule names. The interpretation of vertex labeling is that for the vertex to be satisfied, it must be possible to satisfy all the formulae in the set together. Each edge in the graph represents a satisfaction step of the formula contained in the starting vertex. These steps correspond to the application of a set of tableau rules. These rules express how the satisfaction of a particular formula (the goal) can be obtained by the satisfaction of its constituent formulae (sub-goals).

3 ACTL* Logic

In this section, we present ACTL* logic that we use to specify dialogue game protocols and to express the properties to be verified (See Fig. 1). This specification will be addressed in Section 4. ACTL* is a simplification of our logic for agent communication [6]. ACTL* extends CTL* by allowing formulae to constrain actions as well as propositions. The difference between ACTL* and CTL* is that the former contains action formulae and two new operators: SC for social commitments and $\therefore$ for arguments. The set of atomic propositions is denoted Γp. The set of action labels is denoted Γa. In what follows we use $p, p_1, p_2, \ldots$ to range over the set of atomic propositions and $\theta, \theta_1, \theta_2, \ldots$ to range over action labels. The syntax of this logic is as follows:

$$\mathcal{S} ::= p | \neg p | \mathcal{S} \wedge \mathcal{S} | \mathcal{S} \vee \mathcal{S} | A\mathcal{P} | E\mathcal{P} | SC(Ag_1, Ag_2, \mathcal{P})$$
$$\mathcal{P} ::= \theta | \neg \theta | \mathcal{S} | \mathcal{P} \wedge \mathcal{P} | \mathcal{P} \vee \mathcal{P} | X\mathcal{P} | \mathcal{P}U\mathcal{P} | \mathcal{P} \therefore \mathcal{P}$$

The formulae generated by $\mathcal{S}$ are called state formulae, while those generated by $\mathcal{P}$ 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*). The formula $SC(Ag_1, Ag_2, \phi)$ means that agent Ag_1 commits towards agent Ag_2 that the path formula ϕ is true. We notice here that agents can commit about path formulae, which is expressive enough. Ag_1 and Ag_2 are respectively called the *debtor* and the *creditor* of the commitment. The formula $\phi_1 \therefore \phi_2$ 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.

Semantically, this logic is interpreted with respect to the model M defined as follows: $M = \langle S_m, Lab_m, Act_m, \xrightarrow{Act_m}, Agt, R_{sc}, s_{m_0} \rangle$ where: S_m is a set of states; $Lab_m : S_m \to 2^{\Gamma p}$ is the labeling state function; Act_m is a set of actions; $\xrightarrow{Act_m} \subseteq S_m \times Act_m \times S_m$ is the transition relation; Agt is a set of communicating agents; $R_{sc} : S_m \times Agt \times Agt \to 2^{\sigma}$ 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 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 $x = s_{m_0} \xrightarrow{\alpha_1} s_{m_1} \xrightarrow{\alpha_2} s_{m_2} \ldots$ where $x \in \sigma$, $s_{m_0}, s_{m_1}, \ldots$ are states and $\alpha_1, \alpha_2, \ldots$ are actions. We write $\theta \trianglerighteq \alpha_i$ where $i \geq 1$ to indicate that the action label θ is a part of the action α_i (the action α_i is composed of several action labels). If not, we write $\theta \ntrianglerighteq \alpha_i$.

A state satisfies $A\phi$ ($E\phi$) if every path (some path) emanating from this state satisfies ϕ. The semantics of ACTL* state formulae is as usual (semantics of CTL*). A state s_m satisfies $SC(Ag_1, Ag_2, \phi)$ if every accessible path to Ag_1 towards Ag_2 from this state using R_{sc} satisfies ϕ. Formally:
$s_m \models_M SC(Ag_1, Ag_2, \phi)$ iff $\forall x \in \sigma,\ x \in R_{sc}(s_m, Ag_1, Ag_2) \Rightarrow x \models_M \phi$.

A path satisfies a state formula if the initial state in the path does. A path x satisfies an action label θ if θ is in the label of the first transition on this path and this path is not a *deadlocked* path. A path is deadlocked if it has no transitions. A path satisfies $\neg\theta$ if either θ is not in the label of the first transition on this path or this path is a deadlocked path. Formally:
$x \models_M \theta$ iff $\theta \trianglerighteq \alpha_1$ and x *is not a deadlocked path*
$x \models_M \neg\theta$ iff $\theta \ntrianglerighteq \alpha_1$ or x *is a deadlocked path*
where the action α_1 is the label of the first transition on the path x.

X represents the next time operator and has the usual semantics when the path is not deadlocked. On a deadlocked path, $X\phi$ holds if the current state satisfies ϕ.

Along a path x, $\phi_1 U \phi_2$ holds if ϕ_1 remains true along this path until ϕ_2 becomes true (strong until). Formally:
$x \models_M \phi_1 U \phi_2$ iff $\exists i \geq 0 : x^i \models_M \phi_2$ and $\forall j < i, x^j \models_M \phi_1$
where $x^i = s_{m_i} \xrightarrow{\alpha_{i+1}} s_{m_{i+1}} \ldots$ is the suffix of the path x starting from the *ith* state.

Along a path x, $\phi_1 \therefore \phi_2$ holds if ϕ_1 is true and at next time if ϕ_1 is true then ϕ_2 is true. Formally:
$x \models_M \phi_1 \therefore \phi_2$ iff $x \models_M \phi_1$ and $x^1 \models_M \phi_1 \Rightarrow \phi_2$.

To specify dialogue game protocols in this logic according to the CAN framework, we use the set Act ($Act \subseteq \mathit{\Gamma a}$) of the actions performed by the agents on social commitments (SC) and on SC contents. The idea behind the CAN framework is that agents communicate by performing actions on SC (for example creating, accepting and challenging SC) and by supporting these actions by argumentation relations (attack, defense, and justification). Such an approach, also called the social approach [23] is considered as an alternative to the private approach based on the agents' mental states like beliefs, desires, and intentions [12]. In our dialogue game protocols, we consider the following CAN-based action formulae that constitutes the set Act:

$Cr(Ag_1, SC(Ag_1, Ag_2, \phi))$ (creation of the commitment by the debtor Ag_1)
$Wit(Ag_1, SC(Ag_1, Ag_2, \phi))$ (withdrawal of the commitment by Ag_1)
$Sat(Ag_1, SC(Ag_1, Ag_2, \phi))$ (satisfaction of the commitment by Ag_1)
$Vio(Ag_1, SC(Ag_1, Ag_2, \phi))$ (violation of the commitment by Ag_1)
$Ac(Ag_2, SC(Ag_1, Ag_2, \phi))$ (acceptance by the creditor Ag_2 of the content ϕ)
$Ref(Ag_2, SC(Ag_1, Ag_2, \phi))$ (refusal by Ag_2 of the content ϕ)
$Ch(Ag_2, SC(Ag_1, Ag_2, \phi))$ (challenge by Ag_2 of the content ϕ)
$At(Ag_2, SC(Ag_1, Ag_2, \phi_1), \phi_2)$ (attack by Ag_2 of the content ϕ_1 using ϕ_2)
$Def(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)$ (defense by Ag_2 of the content ϕ_1 using ϕ_2)
$Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)$ (justification by Ag_2 of the content ϕ_1 using ϕ_2)

4 Dialogue Game Protocols as Transition Systems

In Section 3, we presented ACTL* logic and CAN-based actions. In this section, we specify the dialogue game protocols to be checked as models for this logic (see Fig. 1). This specification uses CAN-based actions and the labels of our tableau rules that we will present in Section 5.1. Dialogue game protocols are specified as a set of rules describing the entry condition, the dynamics and the exit condition [5]. These rules can be specified as CAN-based actions.

We propose to define dialogue game protocols as TS. The purpose of these TS is to describe not only the sequence of the allowed actions (classical TS), but also the tableau rules-based decomposition of these actions (Section 5.1). The states of these systems are sub-TS (that we call decomposition TS) describing the tableau rules-based decomposition of the actions labeling the entry transitions. Defining TS in such a way allows us to verify: (1) The correctness of the protocol (if the model of the protocol satisfies the properties that the protocol should specify); (2) The compliance to the decomposition semantics of the communicative actions (if the specification of the protocol respects the decomposition semantics). In Section 5 we propose a model checking procedure in order to verify both (1) and (2) at the same time. The definition of the TS of dialogue game protocols is given by the following definitions:

Definition 1 (Decomposition TS). *A decomposition TS T' describing the tableau-rules-based decomposition semantics of a CAN based-action formula is a 7-tuple $\langle S', Lab', F, L', R, \xrightarrow{R}, s'_0 \rangle$ where: S' is a set of states; $Lab' : S' \rightarrow 2^{\Gamma p}$ is the labeling state function; F is a set of ACTL* formulae; $L' : S' \rightarrow 2^F$ is a function associating a set of formulae to a state; $R \in \{\wedge, \vee, \neg, <\!\!\equiv\!\!>, X, SC_{Ag}\}$ is a tableau rule label (without the rules for CAN-based action formulae) (see Section 5.1); $\xrightarrow{R} \subseteq S' \times R \times S'$ is the transition relation; s'_0 is the start state.*

Intuitively, states S' contain the sub-formulae of the CAN-based action formulae, and the transitions are labeled by operators associated with the formula of the starting state. Decomposition TS enable us to describe the decomposition 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 of $L'(S'_i)$.

Definition 2 (TS for Dialogue Game Protocols). *A TS T for a dialogue game protocol is a 7-tuple $\langle S, Lab, \wp, L, Act, \xrightarrow{Act}, s_0 \rangle$ where: S is a set of states; $Lab : S \rightarrow 2^{\Gamma p}$ is the labeling state function; $\wp$ is a set of decomposition TS with $\varepsilon \in \wp$ is the empty decomposition TS; $L : S \rightarrow \wp$ is the function associating to a state $s \in S$ a decomposition TS $T' \in \wp$ describing the tableau-based decomposition of the CAN-based action labeling the entry transition; Act is the set of CAN-based actions; $\xrightarrow{Act} \subseteq S \times Act \times S$ is the transition relation; s_0 is the start state with $L(s_0) = \varepsilon$.*

We write $s \xrightarrow{\bullet} s'$ instead of $< s, \bullet, s' > \in \xrightarrow{Act}$ where $\bullet \in Act$. Fig. 2 illustrates a part of a TS for a dialogue game protocol. According to this protocol, if Ag_1

plays a creation game ($a1$), Ag_2 can, for instance, play a challenge game ($a2$). Thereafter, Ag_1 must plays a justification game ($a3$), etc.

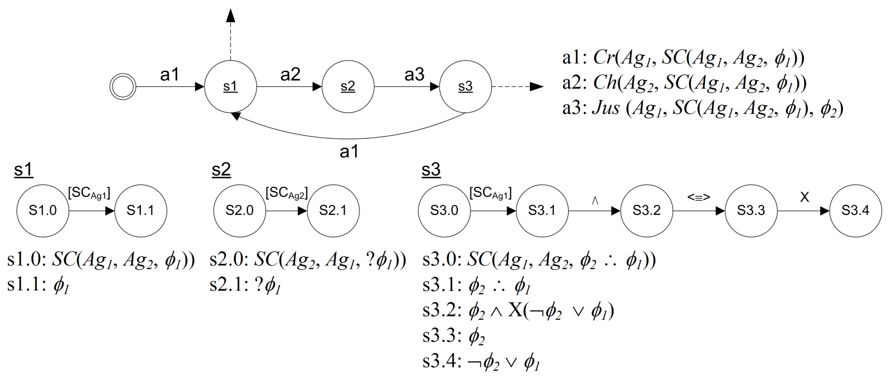
a1
a2
a3
s1
s2
s3
a1
a1: $Cr(Ag_1, SC(Ag_1, Ag_2, \phi_1))$
a2: $Ch(Ag_2, SC(Ag_1, Ag_2, \phi_1))$
a3: $Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)$
s1
S1.0
[SC$_{Ag1}$]
S1.1
s2
S2.0
[SC$_{Ag2}$]
S2.1
s3
S3.0
[SC$_{Ag1}$]
S3.1
$\wedge$
S3.2
<≡>
S3.3
X
S3.4
s1.0: $SC(Ag_1, Ag_2, \phi_1)$)
s1.1: ϕ_1
s2.0: $SC(Ag_2, Ag_1, ?\phi_1)$)
s2.1: $?\phi_1$
s3.0: $SC(Ag_1, Ag_2, \phi_2 \therefore \phi_1)$)
s3.1: $\phi_2 \therefore \phi_1$
s3.2: $\phi_2 \wedge X(\neg\phi_2 \vee \phi_1)$
s3.3: ϕ_2
s3.4: $\neg\phi_2 \vee \phi_1$

Fig. 2. A part of a transition system for a dialogue game protocol

States $S1$, $S2$, and $S3$ are decomposition TS associated respectively with creation, challenge, and justification actions. For example, for the creation action ($S1$), the first state ($s1.0$) is associated with the SC formula according to the rule $R6$ (Table 1, Section 5.1). The next state is associated with the SC content according to the rule $R16$ (Table 1). The transition is labeled with the label of this rule. An example of the properties to be verified in this protocol is:

$$AG(Ch(Ag_2, SC(Ag_1, Ag_2, \phi_1)) \Rightarrow F(Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2))) \qquad (1)$$

This property says that in all paths (A) globally (G), if an agent Ag_2 challenges (Ch) the content of a SC made by an agent Ag_1, then in the future (F), Ag_1 justifies (Jus) the content of its SC. In the rest of this paper, we refer to this formula as **Formula 1**.

5 Verification of Dialogue Game Protocols

In previous sections, we presented the elements needed for the verification of dialogue game protocols: the logic, the CAN-based actions, and the specification of dialogue game protocols. In this section, we present the verification technique. This technique is based upon (1) the tableau rules associated with ACTL* logic (Section 5.1); (2) the ABTA for ACTL* logic (Section 5.2); and (3) the translation of the property to be verified to an ABTA (Section 5.3) (see Fig. 1). This translation is the step 1 of Fig. 1. The step 2 which is the construct of the product graph of the model and the ABTA is addressed in Section 5.4. Finally, the model checking algorithm acting on the product graph (step 3) is presented in Section 5.5. Examples illustrating each step are also presented.

5.1 Tableau Rules for ACTL* and CAN-based Action Formulae

In this section, we present the tableau rules that we use to translate the ACTL* property to be verified to an ABTA (see Fig. 1). The definition of ABTA and the translation procedure will be presented in Section 5.2 and 5.3. The tableau rules allow us to build the ABTA representing the formula to be verified. These rules [11] are specified in terms of the decomposition of formulae to sub-formulae. They enable us to define top-down proof systems. The idea is: given a formula (the top part of the rule), we apply a tableau rule and determine the sub-formulae (the down part of the rule) to be proven (see Section 2). 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 ABTA constructed from the given formula (Section 5.2). These rules are presented in Table 1. In these rules, $\varPhi$ is any set of path formulae. The symbol ”,” indicates a conjunction. For example, $E(\varPhi, \psi)$ means that, there is a path along which the set of path formulae $\varPhi$ and the state formula ψ are true. Adding the set $\varPhi$ to these rules allows us to deal with any form of formulae written under the form of any set of path formulae and a formula of our logic. We also recall that 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.

Table 1. Tableau rules

$R1 \wedge : \frac{\psi_1 \wedge \psi_2}{\psi_1 \psi_2}$	$R2 \vee : \frac{\psi_1 \vee \psi_2}{\psi_1 \psi_2}$	$R3 \vee : \frac{E(\psi)}{\psi}$	$R4 \neg : \frac{\neg \psi}{\psi}$	$R5 \neg : \frac{A(\varPhi)}{E(\neg \varPhi)}$
$R6 <Cr> : \frac{E(\varPhi, Cr(Ag_1, SC(Ag_1, Ag_2, \phi)))}{E(\varPhi, SC(Ag_1, Ag_2, \phi))}$	$R11 < Ac > : \frac{E(\varPhi, Ac(Ag_2, SC(Ag_1, Ag_2, \phi)))}{E(\varPhi, SC(Ag_2, Ag_1, \phi))}$			
$R7 <Wit> : \frac{E(\varPhi, Wit(Ag_1, SC(Ag_1, Ag_2, \phi)))}{E(\varPhi, \neg SC(Ag_1, Ag_2, \phi))}$	$R12 < Ref > : \frac{E(\varPhi, Ref(Ag_2, SC(Ag_1, Ag_2, \phi)))}{E(\varPhi, SC(Ag_2, Ag_1, \neg \phi))}$			
$R8 <Sat_{Ag_1}> : \frac{E(\varPhi, Sat(Ag_1, SC(Ag_1, Ag_2, \phi)))}{E(\varPhi, \phi)}$	$R13 <Jus> : \frac{E(\varPhi, Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2))}{E(\varPhi, SC(Ag_1, Ag_2, \phi_2 \therefore \phi_1))}$			
$R9 <Vio_{Ag_1}> : \frac{E(\varPhi, Vio(Ag_1, SC(Ag_1, Ag_2, \phi)))}{E(\varPhi, \neg \phi)}$	$R14 <At> : \frac{E(\varPhi, At(Ag_2, SC(Ag_1, Ag_2, \phi_1), \phi_2))}{E(\varPhi, SC(Ag_2, Ag_1, \phi_2 \therefore \neg \phi_1))}$			
$R10 <Ch> : \frac{E(\varPhi, Ch(Ag_2, SC(Ag_1, Ag_2, \phi)))}{E(\varPhi, SC(Ag_2, Ag_1, ?\phi))}$	$R15 <Def> : \frac{E(\varPhi, Def(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2))}{E(\varPhi, SC(Ag_1, Ag_2, \phi_2 \therefore \phi_1))}$			
$R16\ [SC_{Ag_1}] : \frac{E(\varPhi, SC(Ag_1, Ag_2, \phi))}{E(\varPhi, \phi)}$	$R17 <\equiv> : \frac{E(\varPhi, \varPsi)}{E(\varPhi) E(\varPsi)}$	$R18 \wedge : \frac{E(\varPhi, \phi_1 \wedge \phi_2)}{E(\varPhi, \phi_1, \phi_2)}$		
$R19 \vee : \frac{E(\varPhi, \phi_1 \vee \phi_2)}{E(\varPhi, \phi_1) E(\varPhi, \phi_2)}$	$R20\ X : \frac{E(\varPhi, X\phi_1, \ldots, X\phi_n)}{E(\varPhi, \phi_1, \ldots, \phi_n)}$	$R21 \wedge : \frac{E(\varPhi, \phi_1 \therefore \phi_2)}{E(\varPhi, \phi_1, X(\neg \phi_1 \vee \phi_2))}$		
$R22 \vee : \frac{E(\varPhi, \phi_1 U \phi_2)}{E(\varPhi, \phi_2) E(\varPhi, \phi_1, X(\phi_1 U \phi_2))}$				

Rule $R1$ 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. $R3$ labeled by "$\vee$" indicates that ψ is the sub-formula to be proved in order to prove that a state satisfies $E(\psi)$. E is the existential path-quantifier. According to $R4$, the formula $\neg\psi$ is satisfied in a state labeled by "$\neg$" if this state has a successor representing ψ. $R5$ is defined in the usual way.

The label "$<Cr>$" ($R6$) is the label associated with the creation action of a social commitment. According to this rule, in order to prove that a state labeled by "$<Cr>$" satisfies $Cr(Ag_1, SC(Ag_1, Ag_2, \phi))$, we have to prove that the child state satisfies the sub-formula $SC(Ag_1, Ag_2, \phi)$. The idea is that by creating a social commitment, this commitment becomes true in the child state. In the model representing the dialogue game protocol, the idea behind the creation action is that by creating a social commitment, this commitment becomes true in the accessible state via the transition labeled by the creation action. The label "$< Wit >$" ($R7$) is the label associated with the withdrawal action of a social commitment. According to this rule, in order to prove that a state labeled by "$<Wit>$" satisfies $Wit(Ag_1, SC(Ag_1, Ag_2, \phi))$, we have to prove that the child state satisfies the sub-formula $\neg SC(Ag_1, Ag_2, \phi)$. Rules $R8$ to $R15$ are defined in the same way. For example, the idea of rule $R11$ is that by accepting a social commitment whose content is ϕ by an agent Ag_2, this agent commits about this content in the child state. In this state, the commitment of Ag_2 becomes true. In rule $R10$, we introduce a syntactical construct "?" to indicate that the debtor Ag_2 does not have in argument supporting ϕ or $\neg\phi$. The idea of this rule is that by challenging a social commitment, Ag_2 commits in the child state that it does not have an argument for or against the content ϕ.

Rule $R16$ indicates that $E(\phi)$ is the sub-formula of $E(SC(Ag_1, Ag_2, \phi))$. Thus, in order to prove that a state labeled by "$[SC_{Ag_1}]$" satisfies the formula $E(SC(Ag_1, Ag_2, \phi))$, we have to prove that the child state satisfies the sub-formula $E(\phi)$. According to the semantics of social commitments (Section 3), the idea of this rule is that if an agent commits about a content along a path, this content is true along this path (we recall that the commitment content is a path formula).

Rules $R17$, $R18$, and $R19$ are straightforward. According to rule $R20$ and in accordance with the semantics of "X", in order to prove that a state labeled with "X" satisfies $E(X\phi)$, we have to prove that the child state satisfies the sub-formula $E(\phi)$. According to $R21$ and in accordance with the semantics of "$\therefore$" (Section 3), in order to prove that a state labeled with " $\wedge$" satisfies $E(\phi_1 \therefore \phi_2)$, we have to prove that the child state satisfies the sub-formula $E(\phi_1 \wedge X(\neg\phi_1 \vee \phi_2))$. This mean that the support is true and next if the support is true then the conclusion is true. Finally, rule $R22$ is defined in accordance with the usual semantics of *until* operator.

5.2 Alternating Büchi Tableau Automata (ABTA) for ACTL*

As a kind of Büchi automata, ABTAs [7] are used in order to prove properties of *infinite* behavior. These automata can be used as an intermediate representation for system properties. Let Γp be the set of atomic propositions and let $\Re$ be a set of tableau rule labels defined as follows: [3]
$\Re = \{\wedge, \vee, \neg\} \cup \Re_{Act} \cup \Re_{\neg Act} \cup \Re_{SC} \cup \Re_{Set}$ where: $\Re_{Act} = \{< Cr >, < Wit >, < Sat_{Ag} >, < Vio_{Ag} >, < Ch >, < Ac >, < Ref >, < Jus >, < At >, < Def >\}$, $\Re_{SC} = \{[SC_{Ag}]\}$, and $\Re_{Set} = \{<\!\equiv\!>, X\}$.
We define ABTAs for ACTL* logic as follows:

Definition 3 (ABTA). *An ABTA for ACTL* is a 5-tuple $\langle Q, l, \rightarrow, q_0, F\rangle$, where: Q is a finite set of states; $l : Q \rightarrow \Gamma p \cup \Re$ 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[4].*

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 TS satisfies a temporal formula. Let us consider the following example. We would like to prove that a state s in a TS satisfies a temporal formula of the form $F_1 \wedge F_2$, where F_1 and F_2 are two formulae. Regardless of the structure of the system, there would be two sub-goals. The first would be to prove that s satisfies F_1, and the second would be to prove that s satisfies F_2. Intuitively, an ABTA for $F_1 \wedge F_2$ would encode this "proof structure" using states for the formulae $F_1 \wedge F_2$, F_1, and F_2. A transition from $F_1 \wedge F_2$ to each of F_1 and F_2 should be added to the ABTA and the labeling of the state for $F_1 \wedge F_2$ being "$\wedge$" 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.

5.3 Translating ACTL* into ABTA (Step 1)

The procedure for translating an ACTL* 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 if states satisfy properties. The tableau is constructed by exhaustively applying the tableau rules presented in Table 1 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

[3] The partition of the set of tableau rule labels is only used for readability and organization reasons

[4] The notion of acceptance condition is related to the notion of accepting run that we define in Section 5.4

at node q. The labeling function l of states is defined as follows. If q does not have any successor, then $l(q) \in \Gamma p$. 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 ACTL* formula p is a maximal proof tree having p as its root and constructed using our tableau rules (see Section 5.1). 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 $<p_0, p_1, \ldots>$, where p_{i+1} is the child of p_i [11].

Example 1. In order to illustrate the translation procedure and the construction of an ABTA from an ACTL* formula, let us consider our formula Formula 1 given in Section 4. Table 2 is the tableau to build for translating Formula 1 into an ABTA. The form of Formula 1 is: $AG(p \Rightarrow q)(\equiv AG(\neg p \vee q))$ (the root of Table 2). The first rule we can apply is $R5$ labeled by $\neg$ in order to transform *all paths* to *exists a path*. We also use the equivalence $(F(p) \equiv \neg G(\neg p))$. We then obtain the child number (2). The next rule we can apply is $R22$ labeled by $\vee$ because F is an abbreviation of U $(F(p) \equiv True\ U\ p)$. Consequently, we obtain two children (3) and (4). From the child (3) we obtain the child (5) by applying the rule $R10$, and from the child (4) we obtain the child (2) by applying the rule $R20$ etc. The ABTA obtained from this tableau is illustrated by Fig. 3. States are labeled by the child's number in the tableau and the label of the applied rule according to Table 2.

Table 2. The tableau of Formula 1

$\neg : AG(\neg Ch(Ag_2, SC(Ag_1, Ag_2, \phi_1)) \vee F(Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(1)**	
$\vee : EF(Ch(Ag_2, SC(Ag_1, Ag_2, \phi_1)) \wedge G(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(2)**	
$<Ch>$: $E(Ch(Ag_2, SC(Ag_1, Ag_2, \phi_1)) \wedge G(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(3)**	$<X>$: $EX(F(Ch(Ag_2, SC(Ag_1, Ag_2, \phi_1)) \wedge G(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2))))$ **(4)**
$[SC_{Ag_2}] : E(SC(Ag_2, Ag_1, ?\phi_1) \wedge G(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(5)**	$EF(Ch(Ag_2, SC(Ag_1, Ag_2, \phi_1)) \wedge G(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(2)**
$<\equiv>$: $E(?\phi_1 \wedge G(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(6)**	
$?\phi_1$ **(7)**	$\vee : E(G(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(8)**
	$<\neg Jus>$: $E(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2), XG(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(9)**
$[SC_{Ag_1}] : E(SC(Ag_1, Ag_2, \phi_1 \therefore \phi_2), XG(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(10)**	
$\wedge : E(\phi_2 \therefore \phi_1, XG(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(11)**	
$<\equiv>$: $E(\phi_2, X(\neg \phi_2 \vee \phi_1), XG(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(12)**	
ϕ_2 **(13)**	$X : E(X(\neg \phi_2 \vee \phi_1), XG(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(14)**
	$<\equiv>$: $E((\neg \phi_2 \vee \phi_1), XG(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(15)**
$\neg \phi_2 \vee \phi_1$ **(16)**	$X : E(XG(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(17)**
	$\vee : E(G(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1), \phi_2)))$ **(8)**

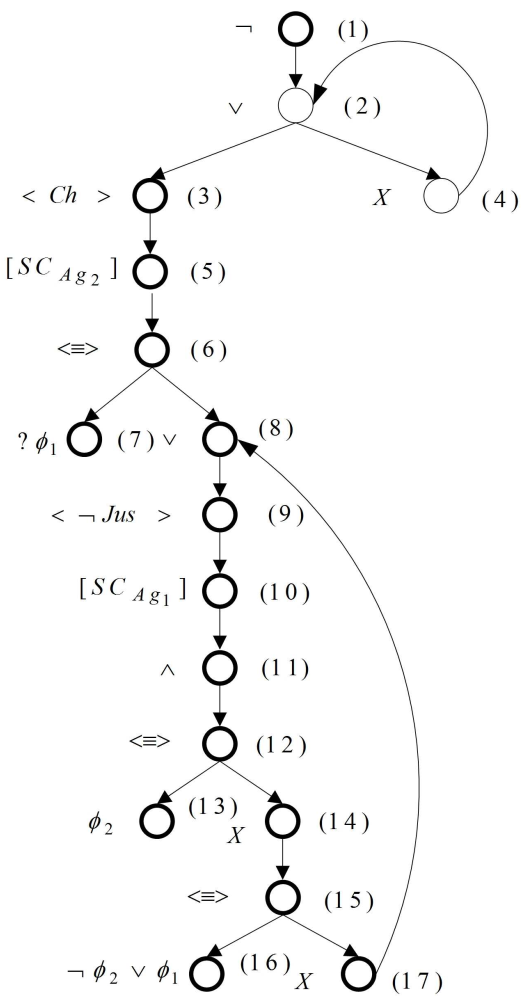

Fig. 3. The ABTA of Formula 1

The termination proof of the translation procedure is based on the finiteness of the tableau. This proof is based on the length of formulae and an ordering relation between these formulae. The proof is detailed in [4].

5.4 Run of an ABTA on a Transition System (Step 2)

Like the automata-based model checking of PLTL, in order to decide about the satisfaction of formulae, we use the notion of the *accepting runs*. In our technique, we need to define the accepting runs of an ABTA on a TS. Firstly, we have to define the notion of the *ABTA's run*. For this reason, we need to introduce two types of nodes: *positive* and *negative*. Intuitively, nodes classified positive are nodes that correspond to a formula without negation, and negative nodes are nodes that correspond to a formula with negation. Definition 4 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.

Definition 4 (Run of an ABTA). *A run of an ABTA $B = \langle Q, l, \rightarrow, q_0, F\rangle$ on a TS $T = \langle S, Lab, \wp, L, Act, \xrightarrow{Act}, s_0\rangle$ is a graph in which the nodes are classified as positive or negative 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 $<q_0, s_0>$.*
2. *For a positive node φ with label $<q, s_i>$:*
   - *(a) If $l(q) = \neg$ and $q \rightarrow q'$, then φ has one negative successor labeled $<q', s_i>$ and vice versa.*
   - *(b) If $l(q) \in \Gamma p$, then φ is a leaf.*
   - *(c) If $l(q) \in \{\wedge, <\!\!\equiv\!\!>\}$ 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 $<q_j, s_i>$ $(1 \leq j \leq m)$.*
   - *(d) If $l(q) = \vee$, then φ has one positive successor φ' labeled by $<q', s_i>$ for some $q' \in \{q' | q \rightarrow q'\}$.*
   - *(e) If $l(q) = X$ and $q \rightarrow q'$ and $\{s' | s_i \xrightarrow{\bullet} s'\} = \{t_1, \ldots, t_m\}$ where $\bullet \in Act$, then φ has positive successors $\varphi_1, \ldots, \varphi_m$ with φ_j labeled by $<q', t_j>$ $(1 \leq j \leq m)$.*
   - *(f) If $l(q) = <\bullet>$ where $\bullet \in Act$ and $q \rightarrow q'$, and $s_i \xrightarrow{\bullet} s_{i+1}$, then φ has one positive successor φ' labeled by $<q', s_{i+1,0}>$ where $s_{i+1,0}$ is the initial state of the decomposition TS of s_{i+1}.*
   - *(g) If $l(q) = <\bullet>$ where $\bullet \in \neg Act$ and $q \rightarrow q'$, and $s_i \xrightarrow{\bullet'} s_{i+1}$ where $\bullet \neq \bullet'$ and $\bullet' \in Act$, then φ has one positive successor φ' labeled by $<q', s_{i+1}>$.*
3. *For a negative node φ labeled by $<q, s_i>$:*
   - *(a) If $l(q) \in \Gamma p$, then φ is a leaf.*
   - *(b) If $l(q) \in \{\vee, <\!\!\equiv\!\!>\}$ and $\{q' | q \rightarrow q'\} = \{q_1, \ldots, q_m\}$, then φ has negative successors $\varphi_1, \ldots, \varphi_m$ with φ_j labeled by $<q_j, s_i>$ $(1 \leq j \leq m)$.*
   - *(c) If $l(q) = \wedge$, then φ has one negative successor φ' labeled by $<q', s_i>$ for some $q' \in \{q' | q \rightarrow q'\}$.*
   - *(d) If $l(q) = X$ and $q \rightarrow q'$ and $\{s' | s_i \xrightarrow{\bullet} s'\} = \{t_1, \ldots, t_m\}$ where $\bullet \in Act$, then φ has negative successors $\varphi_1, \ldots, \varphi_m$ with φ_j labeled by $<q', t_j>$ $(1 \leq j \leq m)$.*
   - *(e) If $l(q) = <\bullet>$ where $\bullet \in Act$ and $q \rightarrow q'$, and $s_i \xrightarrow{\bullet} s_{i+1}$, then φ has one negative successor φ' labeled by $<q', s_{i+1,0}>$ where $s_{i+1,0}$ is the initial state of the decomposition TS of s_{i+1}.*
   - *(f) If $l(q) = <\bullet>$ where $\bullet \in \neg Act$ and $q \rightarrow q'$, and $s_i \xrightarrow{\bullet'} s_{i+1}$ where $\bullet \neq \bullet'$ and $\bullet' \in Act$, then φ has one negative successor φ' labeled by $<q', s_{i+1}>$.*
4. *Otherwise, for a positive (negative) node φ labeled by $<q, s_{i,j}>$:*
   - *(a) If $l(q) = <\!\!\equiv\!\!>$ and $\{q' | q \rightarrow q'\} = \{q_1, q_2\}$ 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 $<q_1, s_{i,j}>$ and one positive (negative) successor φ'' labeled by $<q_2, s_{i,j+1}>$.*

(b) If $l(q) = <\equiv>$ and $\{q'|q \rightarrow q'\} = \{q_1, q_2\}$ such that q_1 is a leaf, and $s_{i,j}$ has no successor, then φ has one positive leaf successor φ' labeled by $<q_1, s_{i,j}>$ and one positive (negative) successor φ'' labeled by $<q_2, s_i>$.

(c) If $l(q) \in \{\wedge, \vee, X, [SC_{Ag}]\}$ and $\{q'|q \rightarrow q'\} = \{q_1\}$, and $s_{i,j} \xrightarrow{r} s_{i,j+1}$ such that $r = l(q)$, then φ has one positive (negative) successor φ' labeled by $<q_1, s_{i,j+1}>$.

The notion of run of an ABTA on a TS is a non-synchronized product graph of the ABTA and the TS (see Fig. 1). This run uses the label of nodes in the ABTA ($l(q)$), the transitions in the ABTA ($q \rightarrow q'$), and the transitions in the TS ($s_i \xrightarrow{\bullet} s_j$). 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 TS (this is the case for example of clauses $2.a, 2.c$, and $2.d$).

The clause $2.a$ 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 $q \rightarrow q'$ in this ABTA, then φ has one negative successor labeled with $<q', s_i>$. In this case we use a transition from the ABTA and we stay in the same state of the TS. 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 $2.c$). However, if the current state of the ABTA is labeled with $\vee$, only one arbitrary transition from the ABTA is used (clause $2.d$). 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. However, in the case of $\vee$ only one sub-formula must be true.

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

1. The current node of the ABTA is labeled with X (which means a next state in the TS). This is the case of clauses $2.e$ and $3.d$. In this case we use all the transitions from the current state s_i to next states of the TS.
2. The current state of the ABTA and a transition from the current state of the TS are labeled with the same action. This is the case of clauses $2.f$ and $3.e$. In this case, the current transition of the ABTA and the transition from the current state s_i of the TS to a state $s_{i+1,0}$ of the associated decomposition TS are used. The idea is to start the parsing of the formula coded in the decomposition TS.
3. The current state of the ABTA and a transition from the current state of the TS are labeled with different actions where the state of the ABTA is labeled with a negative formula. This is the case of clauses $2.g$ and $3.f$. In this case, the formula is satisfied. Consequently, the current transition of the ABTA and the transition from the current state s_i of the TS to a next state s_{i+1} are used. Finally, clauses $4.a$, $4.b$, and $4.c$ deal with the case of verifying the structure of the commitment formulae in the sub-TS. In these clauses, transitions $s_{i,j} \xrightarrow{r} s_{i,j+1}$ 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).

Example 2. Fig. 4 illustrates an example of the run of an ABTA. This figure illustrates a part of the automaton $B_{\otimes}$ resulting from the product of the TS of Fig. 2 and the ABTA of Fig. 3. According to the clause 1 (Definition 4), the root is a positive node and it is labeled by $<\neg, s_0>$ because the label of the ABTA's root is $\neg$ (Fig. 3). Consequently, according to the clause $2.a$, the successor is a negative node and it is labeled by $<\vee, s_0>$. According to the clause $3.b$, the second node has two negative successors labeled by $<<Ch>, s_0>$ and $<X, s_0>$ etc.

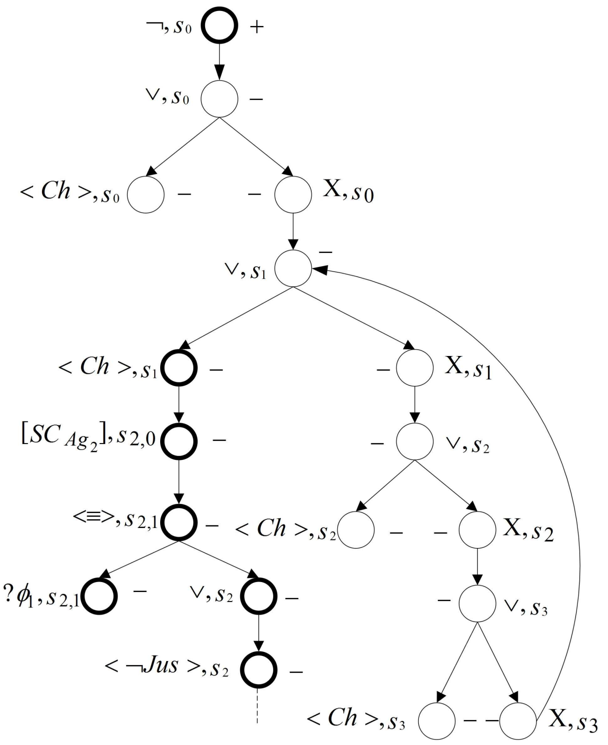

Fig. 4. An example of an ABTA's run

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. Now we can define the notion of accepting runs (or successful runs). Let $p \in \Gamma p$ and let s_i be a state in a TS T. Then $s_i \models_T p$ iff $p \in Lab(s_i)$ and $s_i \models_T \neg p$ iff $p \notin Lab(s_i)$. Let $s_{i,j}$ be a state in a decomposition TS of a TS T. Then $s_{i,j} \models_T p$ iff $p \in Lab'(s_{i,j})$ and $s_{i,j} \models_T \neg p$ iff $p \notin Lab'(s_{i,j})$.

Definition 5 (Successful Run). *Let r be a run of an ABTA $B = \langle Q, l, \rightarrow, q_0, F\rangle$ on a TS $T = \langle S, Lab, \wp, L, Act, \xrightarrow{Act}, s_0\rangle$. 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 $<q, s_i>$ is successful iff $s_i \models_T l(q)$ or $l(q) = <\bullet>$ where $\bullet \in Act$ and there is no s_j such that $s_i \xrightarrow{\bullet} s_j$.*
2. *A positive leaf labeled by $<q, s_{i,j}>$ is successful iff $s_{i,j} \models_T l(q)$*
3. *A negative leaf labeled by $<q, s_i>$ is successful iff $s_i \models_T \neg l(q)$ or $l(q) = <\bullet>$ where $\bullet \in Act$ and there is no s_j such that $s_i \xrightarrow{\bullet} s_j$.*
4. *A negative leaf labeled by $<q, s_{i,j}>$ is successful iff $s_{i,j} \models_T \neg l(q)$*

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 $<q, s>$ such that $l(q) = <\bullet>$ where $\bullet \in Act$ and there is no s' such that $s \xrightarrow{\bullet} s'$ is considered a successful leaf because we can not 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 TS. 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.

An ABTA B accepts a TS T iff there exists a successful run of B on T. 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 6 (Acceptance States). *Let q be a state in an ABTA B and Q the set of all states. Suppose $\phi = \phi_1 U \phi_2 \in q$ [5]. We define the set F_ϕ as follows: $F_\phi = \{q' \in Q | (\phi \notin q'$ and $X\phi \notin q')$ or $\phi_2 \in q'\}$. The acceptance set F is defined as follows: $F = \{F_\phi | \phi = \phi_1 U \phi_2$ 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 4, there is a transition from a state containing ϕ to a state containing $X\phi$ and vice versa. Therefore, according to Definition 5, there is a successful run in the ABTA B. However, we can not 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 can not 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.

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

Example 3. In order to compute the acceptance states of the ABTA of Fig. 3, we use the formula associated to the child number (2) in Table 2:

$$F(Ch(Ag_2, SC(Ag_1, Ag_2, \phi_1)) \wedge G(\neg Jus(Ag_1, SC(Ag_1, Ag_2, \phi_1)\phi_2)))$$

We consider this formula, denoted ϕ, instead of the root's formula because its form is $E(\phi)$ (see Section 5.3). Consequently, state (1) and states from (3) to (17) are the acceptance states according to Definition 6. For example, state (1) is an acceptance state because ϕ and $X\phi$ are not in this state, and state (3) is an acceptance state because ϕ_2 is in this state. 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 Formula 1 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 Formula 1 because there is a challenge action (state (3)), and globally no justification action of the content of the challenged commitment (state (9)).

5.5 Model Checking Algorithm (Step 3)

Our model checking algorithm for verifying that a dialogue game protocol satisfies a given property and that it respects the decomposition semantics of the underlying communicative acts is inspired by the procedure proposed by [7]. Like the algorithm proposed by [14], our algorithm explores the product graph of an ABTA representing an ACLT* formula and a TS for a dialogue game protocol. This algorithm is on-the-fly (or local) algorithm that consists of checking if a TS is accepted by an ABTA. This ABTA-based model checking is reduced to the emptiness of the Büchi automata [31]. The emptiness problem of automata is to decide, given an automaton A, whether its language $L(A)$ is empty. The language $L(A)$ is the set of words accepted by A.

Let $T = \langle S, Lab, \wp, L, Act, \xrightarrow{Act}, s_0 \rangle$ be a TS for a dialogue game and let $B = \langle Q, l, \rightarrow, q_0, F \rangle$ be an ABTA for ACTL*. 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 $<q, s>$ and $<q', s'>$ 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 [7, 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 or negative). In addition, 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 directly works on the product graph.

The idea of this algorithm is to construct the product graph while exploring it. The construction procedure is directly obtained from Definition 4. The algorithm uses the label of nodes in the ABTA, and the transitions in the product graph obtained from the TS and the ABTA as explained in Definition 4. 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 explores recursively the successors of this node. If this node is labeled by " $\wedge$", and signed positively, then it returns false if one of the successors is false. However, if the node is signed negatively, it returns false if all the successors are false. A dual treatment is applied when the node is labeled by " $\vee$ ".

Example 4. In order to check if the language of the automaton illustrated by Fig. 4 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 the ABTA of Fig. 3. If such a path exists, then we conclude that Formula 1 is not satisfied by the TS of Fig. 2. 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 $(<Ch>, s_0)$ is successful since there is no state s_i such that $s_0 \xrightarrow{Ch} s_i$. Therefore, the TS of Fig. 2 is accepted by the ABTA of Formula 1. Consequently, this TS satisfies Formula 1 and respects its decomposition semantics.

Soundness and completeness of our model checking method are stated by the following theorem.

Theorem 1 (Soundness and Completeness). *Let ψ be a ACTL* formula and B_ψ the ABTA obtained by the translation procedure described above, and let $T = \langle S, Lab, \wp, L, Act, \xrightarrow{Act}, s_0\rangle$ be a TS that represents a dialogue game protocol. Then, $s_0 \models_T \psi$ iff T is accepted by B_ψ.*

Proof. (**Direction $\Rightarrow$**). To prove that T is accepted by B_ψ, we have to prove that there exists a run r of B_ψ on T such that all leaves and all infinite paths in the run are successful. Let us assume that $s_0 \models_T \psi$. First, let us suppose that there exists a leaf $<q, s>$ in r such that $s \models \neg l(q)$. Since the application of tableau rules does not change the satisfaction of formulae, it follows from Definition 4 that $s_0 \models_T \neg\psi$ which contradicts our assumption.
Now, we will prove that all infinite paths are successful. The proof proceeds by contradiction. ψ is a state formula that we can write under the form $E\Phi$, where Φ is a set of path formulae. Let us assume that there exists an unsuccessful infinite path x_r in r and prove that $x_T \models_T \neg\Phi$ where x_T is the path in T that corresponds to x_r (x_r is the product of B_ψ and T). The fact that x_r is infinite implies that $R22$ occurs at infinitely many positions in x_r. Because x_r is unsuccessful, $\exists \phi_1, \phi_2, q_i$ such that $\phi_1 U \phi_2 \in q_i$ and $\forall j \geq i$ we have $\phi_2 \notin q_j$. When this formula appears in the ABTA at the position q_i, we have $l(q_i) = \vee$.

Thus, according to Definition 4 and the form of $R22$, the current node φ_1 of r labeled by $<q_i, s>$ has one successor φ_1 labeled by $<q_{i+1}, s>$ with $\phi_1 U \phi_2 \in q_i$ and $\{\phi_1, X(\phi_1 U \phi_2)\} \subseteq q_{i+1}$. Therefore, $l(q_{i+1}) = \wedge$, and φ_2 has a successor φ_3 labeled by $< q_{i+2}, s >$ with $X(\phi_1 U \phi_2) \in q_{i+2}$. Using $R20$ and the fact that $l(q_{i+2}) = X$, the successor φ_4 of φ_3 is labeled by $<q_{i+3}, s'>$ with $\phi_1 U \phi_2 \in q_{i+3}$ and s' is a successor of s. This process will be repeated infinitely since the path is unsuccessful. It follows that there is no s in T such that $s \models_T \phi_2$. Thus, according to the semantics of U, there is no s in T such that $s \models_T \phi_1 U \phi_2$. Therefore, $x_T \models_T \neg\Phi$.
(**Direction** $\Leftarrow$). The proof proceeds by an inductive construction of x_r and an analysis of the different tableau rules. A detailed proof of this theorem is presented in [4].

6 Related Work and Conclusion

The verification problem has recently begun to find a significant audience in the MAS community. Rao and Georgeff [26] have proposed an adaptation of CTL and CTL* model checking to verify BDI (beliefs, desires and intentions) logics. van der Hoek and Wooldridge [30] have proposed some techniques in order to reduce the model checking of temporal epistemic properties of MAS to the model checking of LTL. Benerecetti and Cimatti [3] have proposed a general approach for model-checking MAS based on CTL together with modalities for BDI attitudes. Wooldridge et al. [33] have presented the translation of the MABLE language for the specification and the verification of MAS into Promela, the language of the SPIN model checker of LTL. Bordini et al. [9, 10] have addressed the problem of verifying MAS specified using the AgentSpeak language. They have showed how programs written in AgentSpeak can be automatically transformed into Promela and into Java, the language of the JPF2 model checker. Penczek and Lomuscio [24] have developed a bounded model checking algorithm for branching time logic for knowledge (CTLK). In a similar way, Raimondi and Lomuscio [25] have implemented an algorithm to verify epistemic CTL properties of MAS using ordered binary decision diagrams. Kacprzak et al. [20] have also investigated the problem of verifying epistemic properties using CTLK by means of an unbounded model checking algorithm. Kacprzak and Penczek [19] have addressed the problem of verifying game-like structures by means of unbounded model checking. There are many differences between all these proposals and the work presented in this paper that we can summarize as follows. First, these proposals are based on BDI and epistemic logics that stress the agents' private mental states. In contrast, our work uses a logic highlighting the public states reflecting the agents' interactions expressed in terms of SC and argumentation relations. Second, our model checking algorithm allows us to verify not only the system's temporal properties but also the action properties. Finally, the technique that we use is based on the tableau method and is different from the techniques used for LTL, CTL and CTL*.

Complementarily, the verification of agent communication protocols has been addressed by some research work. Endriss et al. [15] have proposed abductive logic-based agents and some means of determining whether or not these agents behave in conformance to agent communication protocols. Baldoni et al. [2] have addressed the problem of verifying that a given protocol implementation using a logical language conforms to its AUML specification. Alberti et al. [1] have considered the problem of verifying on the fly the compliance of the agents' behavior to protocols specified using a logic-based framework. These approaches are different from our proposal in the sense that they are not based on model checking techniques and they do not address the problem of verifying if a protocol satisfies given properties.

Huget and Wooldridge [18] have used a variation of the MABLE language to define a semantics of agent communication and have showed that the compliance to this semantics can be reduced to a model checking problem. Walton [32] has applied model checking techniques in order to verify the correctness of protocol communication using the SPIN model checker. The model checking techniques used by these two proposals are based on LTL, whereas our technique is based on ACTL*. In addition, our approach is based on a new algorithm and not on the translation of the specification language to the languages of existing model checkers. Unlike these two proposals, our technique allows us to simultaneously verify the correctness of protocols and the agents' conformance to the semantics.

Recently, Giordano et al. [17] have addressed the problem of specifying and verifying agent interaction protocols using a Dynamic Linear Time Temporal Logic (DLTL). The authors have addressed three kinds of verification problems: 1) the compliance of a protocol execution to its specification; 2) the satisfaction of a property in the protocol; 3) the compliance of agents to the protocol. They have showed that these problems can be solved by model checking DLTL. This model checking technique uses a tableau-based algorithm for obtaining a Büchi automaton from a formula in DLTL. Although this work is close to our proposal, there are four main differences between these two approaches: (1) The protocols we dealt with are dialogue game protocols specified using actions that agents apply on SC. However, the protocols used in [17] are abstract protocols specified in terms of the effects of communicative actions and some precondition laws. (2) The model checking technique proposed in [17] uses classical Büchi automaton that is constructed using a tableau-like procedure and propositional rules. Our technique is different because it is based on ABTA and not on traditional Büchi automaton. In addition, the construction of this automaton uses proof rules and not propositional rules. (3) Our approach is based not only on SC like [17], but also on an argumentation theory. Consequently, our protocols are more suitable for MAS because agents can make decisions using their argumentation systems. (4) The dynamic part in our logic is reflected by an action theory, whereas in DLTL, the dynamic part is represented by regular programs.

The contribution of this paper is the proposition of a new verification technique for dialogue game protocols. Our model checking technique allows us to verify both the correctness of the protocols and the agents' compliance with the decomposition semantics of the communicative acts. This technique uses a combination of an automata-based and a tableau-based algorithm to verify temporal and action specifications. The formal properties to be verified are expressed in our ACTL* logic and translated to ABTA using tableau rules. Our model checking algorithm that works on a product graph is an efficient on-the-fly procedure.

As an extension to this work, we intend to use this tableau-based technique to verify MAS specifications and the conformance of agents with these specifications. Another interesting direction for future work is to extend the technique and the logic in order to consider the epistemic properties. Finally, we plan to use this technique to specify and verify agents' trust policies.

Acknowledgements We would like to thank the three anonymous reviewers for their interesting comments, questions, and suggestions. We are also grateful to Josée Desharnais, Yves lespérance, Mohamed Aoun-Allah and Mohamed Mbarki for their suggestions.

References

1. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P.: Compliance verification of agent interaction: a logic-based tool. Proc. of the European Meeting on Cybernetics and Systems Research, Vol. II (2004) 570–575
2. Baldoni, M., Baroglio, C., Martelli, A., Patti, V., and Schifanella, C.: Verifying protocol conformance for logic-based communicating agents. Computational Logic in Multi-Agent Systems, LNAI 3487 (2004) 196–212
3. Benerecetti, M. and Cimatti, A.: Symbolic model checking for multi-agent systems. Proc. of the International Workshop on Model Checking and AI (2002) 1–8
4. Bentahar, J.: A pragmatic and semantic unified framework for agent communication. PhD Thesis, Laval University, Canada May (2005)
5. Bentahar, J., Moulin, B., and Chaib-draa, B.: A persuasion dialogue game based on commitments and arguments. Proc. of the International Workshop on Argumentation in Multi-Agent Systems (2004) 148–164
6. Bentahar, J., Moulin, B., Meyer, J-J, Ch., and Chaib-draa, B.: A logical model for commitment and argument network for agent communication. Proc. of the International Joint Conference on AAMAS (2004) 792–799
7. Bhat, G., Cleaveland, R., and Groce, A.: Efficient model checking via Büchi tableau automata. Computer-Aided Verification, LNCS 2102 (2001) 38–52
8. Bhat, G., Cleaveland, R., and Grumberg, O.: Efficient on-the-fly model checking for CTL*. The IEEE Symposium on Logics in Computer Science (1995) 388–397
9. Bordini, R.H, Fisher, M., Pardavila C., and Wooldridge, M.: Model checking AgentSpeak. Proc. of the International Conference on AAMAS (2003) 409–419
10. Bordini, R.H., Visser, W., Fisher, M., Pardavila, C., and Wooldridge, M.: Model checking multi-agent programs with CASP. Computer-Aided Verification, 2725 (2003) 110–113
11. Cleaveland, R.: Tableau-based model checking in the propositional mu-calculus. Acta Informatica, 27(8) (1990) 725–747
12. Cohen, P.R., and Levesque, H.J.: Persistence, intentions and commitment. Intentions in Communication, MIT Press, (1990) 33–69

13. Colombetti, M.: A commitment-based approach to agent speech acts and conversations. Proc. of the International Autonomous Agent Workshop on Conversational Policies (2000) 21–29
14. Courcoubetis, C., Vardi, M.Y., Wolper, P., and Yannakakis, M.: Memory efficient algorithms for verification of temporal properties. Formal Methods in System Design, vol. 1 (1992) 275–288
15. Endriss, U., Maudet, N., Sadri, F., and Toni, F.: Protocol conformance for logic-based agents. Proc. of the International Joint Conference on AI (2003) 679–684
16. Fornara, N. and Colombetti, M.: Operational specification of a commitment-based agent communication language. Proc. of the International Joint Conference on AAMS (2002), 535–542
17. Giordano, L., Martelli, A., and Schwind, C.: Verifying communicating agents by model checking in a temporal action logic. Logics in Artificial Intelligence, LNAI 3229 (2004) 57–69
18. Huget, M.P and Wooldridge, M.: Model checking for ACL compliance verification. Advances in Agent Communication. LNAI 2922 (2004) 75–90
19. Kacprzak, M. and Penczek, W.: Unbounded model checking for alternating-time temporal logic. The International Joint Conference on AAMAS (2004) 646–653
20. Kacprzak, M., Lomuscio, A., and Penczek, W.: Verification of multiagent systems via unbounded model checking. Proc. of the International Joint Conference on AAMAS, (2004) 638–645
21. Maudet, N., and Chaib-draa, B.: Commitment-based and dialogue-game based protocols, new trends in agent communication languages. Knowledge Engineering Review, 17(2) (2002) 157–179
22. McBurney, P. and Parsons, S.: Games that agents play: A formal framework for dialogues between autonomous agents. Journal of Logic, Language, and Information, (11)3 (2002) 315–334
23. Moulin, B.: The social dimension of interactions in multi-agent systems. Agent and Multi-Agent Systems, Formalisms, Methodologies and Applications. Artificial Intelligence, 1441 (1998) 109–122
24. Penczek, W. and Lomuscio, A.: Verifying epistemic properties of multi-agent systems via model checking. Fundamenta Informaticae, 55(2) (2003) 167–185
25. Raimondi, F. and Lomuscio, A.: Verification of multiagent systems via ordered binary decision diagrams: an algorithm and its implementation. Proc. of the International Joint Conference on AAMAS (2004) 630–637
26. Rao, A.S. and Georgeff, M.P.: A model-theoretic approach to the verification of situated reasoning systems. Proc. of IJCAI (1993) 318–324
27. Sadri, F. Toni, F., and Torroni, P.: Dialogues for negotiation: agent varieties and dialogue sequences, Proc. of the International workshop on Agents, Theories, Architectures and Languages, LNAI 2333 (2001) 405–421
28. Singh, M.P.: Agent communication languages: rethinking the principles. IEEE Computer, 31(12) (1998) 40–47
29. Stirling, C. and Walker, D.: Local model checking in the modal Mu-Calculus. LNCS 354 (1989) 369–383
30. van der Hoek, W. and Wooldridge, M.: Model checking knowledge and time. In Model Checking Software, LNCS 2318 (2002) 95–111
31. Vardi, M. and Wolper, P.: An automata-theoretic approach to automatic program verification. Symposium on Logic in Computer Science (1986) 332–344
32. Walton, D.C.: Model checking agent dialogues. Declarative Agent Languages and Technologies, LNAI 3476 (2005) 132–147

33. Wooldridge, M., Fisher, M., Huget, M.P., and Parsons. S.: Model checking multi-agent systems with MABLE. Proc. of the International Joint Conference on AA-MAS (2002) 952–959