A NEW MODEL CHECKING APPROACH FOR VERIFYING AGENT COMMUNICATION PROTOCOLS

Jamal Bentahar
Laval University,
Dept. of Computer Science
Québec, Canada
e-mail: jamal.bentahar@ift.ulaval.ca

Bernard Moulin
Laval University,
Dept. of Computer Science
Québec, Canada
e-mail: bernard.moulin@ift.ulaval.ca

John-Jules Ch. Meyer
Utrecht University,
Dept. of Computer Science
Utrecht, The Netherlands
e-mail: jj@cs.uu.nl

Abstract

In this paper, we propose a new model checking algorithm for verifying dialogue game protocols (DGP) for multi-agent communication. These protocols are specified as transition systems in which transitions are labeled with communicative acts. Our underlying logic (SCCTL) used to specify the properties to be verified extends CTL* by adding action formulae. the actions we deal with are actions that agents perform on social commitments when communicating. The verification method is based on the translation of SCCTL* formulae into a variant of alternating tree automata called Alternating Büchi Tableau Automata (ABTA). Our algorithm explores the product graph of the protocol and the ABTA representing the formula to be verified. The nodes of the product graph are signed according to the type of the formula (with or without negation). We propose a set of tableau inference rules for specifying the translation procedure. The efficiency of our algorithm is due to the fact that it uses only one depth-first search instead of two. Our algorithm explores directly the product graph using the sign of the nodes. This algorithm is an on-the-fly efficient algorithm.*

Keywords — *Multi-agent communication; model checking; temporal logic.*

1 Introduction

Multi-agent technology is being increasingly used in several application domains: e-commerce, simulation, distributed collaborative systems, etc. It is widely recognized that the success of this new technology is primarily based on the ability of agents to communicate. Recent years have seen an increasing interest in agent communication in order to allow agents to flexibly and autonomously communicate. Thus, several Dialogue Game Protocols (DGP) based on the philosophy of language have been proposed for specifying agent interactions [3], [11], [12]. DGP are interaction games in which each agent plays a move in turn by performing utterances according to a predefined set of rules.

Recently, verifying multi-agent systems using model checking approaches has become an attractive field of research [6], [10]. In these approaches, the system is represented by a finite model M and the specification is represented by a formula ϕ using an appropriate logic. The verification method consists of computing if the model M satisfies ϕ. In the domain of agent communication, this is a new and vast field of research [1], [9].

In this paper, we propose a new efficient model checking algorithm based on tableau inference rules for verifying DGP. These protocols are specified as transition systems in which transitions are labeled with communicative acts. These acts are defined as actions performed by agents on social commitments, for example, creating, accepting, or challenging social commitments [3]. To specify DGP and the properties to be verified, we define SCCTL* logic, which extends CTL* by adding formulae representing actions that agents perform on social commitments. The verification method is based on the translation of formulae into a variant of alternating tree automata called Alternating Büchi Tableau Automata (ABTA) [5]. 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 DGP and the ABTA representing the property to be verified using the sign of the nodes. To our knowledge, until now there is no work that addressed the verification problem of DGP excepted our first proposal [4]. Indeed, the contribution of this paper is an efficient algorithm for model checking DGP using a new logic SCCTL* and tableau rules.

2 Dialogue Game Protocols

We specify DGP to be checked as formal models associated to our SCCTL* logic (see Section 3). These models are action-labeled transition systems. The purpose of DGP is to describe the sequence of the allowed communicative acts. According to our framework [2], these acts are actions that agents perform on social commitments. In what follows the set of atomic propositions is denoted Γp.

Definition 1 (DGP). *A DGP T is a 5-tuple $\langle S, Lab, Act, \xrightarrow{Act}, s_0\rangle$ where: S is a set of states; $Lab : S \rightarrow 2^{\Gamma p}$ is the labeling state function; Act is the set of allowed communicative acts; $\xrightarrow{Act} \subseteq S \times Act \times S$ is the transition relation; s_0 is the start state.*

We write $s \xrightarrow{a} s'$ instead of $< s, a, s' > \in \xrightarrow{Act}$ where $a \in Act$.

3 A Logic for DGP

In this section, we present SCCTL* logic that we use to specify the properties to be verified. SCCTL* extends CTL* by adding social commitment and action formulae. In what follows we use $p, p_1, p_2, \ldots$ to range over the set of atomic propositions. The syntax of this logic is as follows:

$\mathcal{S}$::=$p|\neg p|\mathcal{S} \wedge \mathcal{S}|\mathcal{S} \vee \mathcal{S}|A\mathcal{P}|E\mathcal{P}$
$\mathcal{P}$::=$\mathcal{S}|\mathcal{P} \wedge \mathcal{P}|\mathcal{P} \vee \mathcal{P}|X\mathcal{P}|X^{-}\mathcal{P}|\mathcal{P}U\mathcal{P}|\mathcal{P}S\mathcal{P}$
$|C(Ag_1, SC(Ag_1, Ag_2, \mathcal{P}))|\neg C(Ag_1, SC(Ag_1, Ag_2, \mathcal{P}))$
$|Act(Ag_i, SC(Ag_1, Ag_2, \mathcal{P}))|\neg Act(Ag_i, SC(Ag_1, Ag_2, \mathcal{P}))$

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* with previous (X^-) and since (S) operators). The formula $C(Ag_1, SC(Ag_1, Ag_2, \phi))$ means that agent Ag_1 commits towards agent Ag_2 that the path formula ϕ is true. The formula $Act(Ag_i, SC(Ag_1, Ag_2, \phi))$ means that agent Ag_i ($i \in \{1, 2\}$) performs an action on the social commitment made by Ag_1 towards Ag_2. The set of actions performed on social commitments are *withdrawal* (Wit), *satisfaction* (Sa), *violation* (Vi), *acceptance* (Ac), *refusal* (Re), and *challenge* (Ch) (see [2] for more details).

Semantics. The formal model M associated to this logic corresponds to the DGP (see Section 2). This model is 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; $Lab_m : S_m \rightarrow 2^{\Gamma p}$ is the labeling state function; Act_m is the set of actions performed on social commitments; $\xrightarrow{Act_m} \subseteq S_m \times Act_m \times S_m$ is the transition relation; Agt is a set of communicating agents; $R_{sc} : S_m \times Agt \times Agt \rightarrow 2^{\sigma}$ 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 social 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 $x^i = s_{m_i} \xrightarrow{\alpha_{i+1}} s_{m_{i+1}} \xrightarrow{\alpha_{i+2}} s_{m_{i+2}} \ldots$ 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 SCCTL* state formulae is as usual (semantics of CTL*). A path satisfies a state formula if the initial state in the path does. A path x^i satisfies $C(Ag_1, SC(Ag_1, Ag_2, \phi))$ if C is in the label of the first transition on this path and if every accessible path to Ag_1 towards Ag_2 from the first state of the path using R_{sc} satisfies ϕ. Formally:

$x^i \models_M C(Ag_1, SC(Ag_1, Ag_2, \phi))$ iff $\alpha_{i+1} = C$
$\wedge \forall x^i \in \sigma,\ x^i \in R_{sc}(s_{m_i}, Ag_1, Ag_2) \Rightarrow x^i \models_M \phi$

A path x^i satisfies $Act(Ag_i, SC(Ag_1, Ag_2, \phi))$ 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:

$x^i \models_M Act(Ag_i, SC(Ag_1, Ag_2, \phi))$ iff $\alpha_{i+1} = Act$
$\wedge\ PC(Ag_1, SC(Ag_1, Ag_2, \phi))$

An example of the properties to be verified in our DGP is:

$$A(C(Ag_1, SC(Ag_1, Ag_2, \phi)) \Rightarrow F(Sa(Ag_1, SC(Ag_1, Ag_2, \phi))))$$

This property says that in all paths (A), if an agent Ag_1 creates a social commitment (C), then in the future (F), Ag_1 satisfies it (Sa).

4 Verifying DGP

4.1 Alternating Büchi Tableau Automata

As a kind of Büchi automata, ABTAs [5] are used in order to prove properties of *infinite* behavior. These automata can be used as an intermediate representation for system properties. Let $\Re$ be a set of tableau rule labels defined as follows: $\Re = \{\wedge, \vee, \neg, <C>, <Act>\}$. We define ABTAs for SCCTL* logic as follows:

Definition 2 (ABTA). *An ABTA for SCCTL* 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.*

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. Let us consider the following example. We would like to prove that a state s satisfies a temporal formula of the form $F_1 \wedge F_2$. 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.

4.2 Model Checking Algorithm

Our model checking algorithm is based on the translation of the property to be verified into an ABTA. The procedure for translating a SCCTL* 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 applying the tableau rules associated to SCCTL*. The tableau rules of the CTL* fragment are as usual. The rules associated to path formulae are:

$$<C>\frac{E(C(Ag_1, SC(Ag_1, Ag_2, \phi)))}{E(\phi)}.$$
$$<Act>\frac{E(Act(Ag_1, SC(Ag_1, Ag_2, \phi)))}{E(\phi)}.$$

The ABTA B can be extracted from the 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' 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 $l(q) \in \Gamma p$. 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. An example of the translation procedure is presented in [2].

Like the algorithm proposed by [7], our algorithm explores the product graph of an ABTA representing a SCCLT* formula and a transition system for a DGP. This algorithm is on-the-fly (or local) algorithm that consists of checking if a transition system is accepted by an ABTA. This ABTA-based model checking is reduced to the emptiness of the Büchi automata [14]. 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 $T = \langle S, Lab, Act, \xrightarrow{Act}, s_0\rangle$ be a DGP and let $B = \langle Q, l, \rightarrow, q_0, F\rangle$ be an ABTA for SCCTL*. 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 ∈ F.

Unlike the algorithms proposed in [5], [7], 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 pseudo-code of this algorithm is given in Algorithm 1. The idea of this algorithm is to construct the product graph while exploring it. The algorithm uses the label of nodes in the ABTA, and the transitions in the product graph obtained from the DGP and the ABTA. 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 " ∧ ", 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 " ∨ ". We note that if the DFS does not explore a false node (i.e. it does not return false), then it returns true. In order to make the algorithm easy to understand, we omit the instructions relative to the addition of nodes in the product graph.

```
DFS(v = (q, s)): boolean {
 if v marked visited {
   if (sign(v) = "+" and not accepting(v))
      or (sign(v) = "-" and accepting(v))
        return false
 }
 else {
  mark v visited
  switch(l(q)) {
    case(∧):
      if s is a leaf return false
      else
        switch(sign(v)) {
          case("+"): for all v'' ∈ {v'/v →_{B⊗} v'}
            if not DFS(v'') return false
          case("-"): for all v'' ∈ {v'/v →_{B⊗} v'}
            if DFS(v'') return true
                return false
        }
    case(∨):
      if s is a leaf return false
      else
        switch(sign(v)) {
          case("+"): for all v'' ∈ {v'/v →_{B⊗} v'}
            if DFS(v'') return true
                return false
          case("-"): for all v'' ∈ {v'/v →_{B⊗} v'}
            if not DFS(v'') return false
        }
    case(<C>, <Act>):
      if s is a leaf return true
      else for the v'' ∈ {v'/v →_{B⊗} v'}
        if not DFS(v'') return false
    case(X):
      if s is a leaf return false
      else for the v'' ∈ {v'/v →_{B⊗} v'}
        if not DFS(v'') return false
  } \\ end of switch(l(q))
 }\\ end of else
return true
}
```

Algorithm 1. Exploring product graph algorithm

Theorem 1. Let ψ be a SCCTL* formula and B_ψ the ABTA obtained by the translation procedure, and let $T = \langle S, Lab, Act, \xrightarrow{Act}, s_0\rangle$ be a DGP. Then $s_0 \models \psi$ iff T is accepted by B_ψ.

The proof of this theorem is presented in [2].

Theorem 2 (Correctness). Let B an ABTA and T a DGP. DFS(q_0, s_0) returns true if and only if T is accepted by B.

Proof.
This theorem follows from Theorem 1. Indeed, DFS returns true if and only if all the leaves are successful, and all the infinite paths are successful. The reason is that DFS returns true if and only if it does not find any unsuccessful leaf and any unsuccessful infinite path.

We conclude this section by discussing the worst-case time complexity of our model checking technique.

Lemma 1. Let ψ be a SCCTL* formula, ant let $B_\psi = \langle Q, l, \rightarrow, q_0, F\rangle$ be the ABTA obtained by the translation procedure. Then $|B_\psi| < 2^{|\psi|}$.

Proof.
From the transition procedure, each formula ψ' 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 ψ' of ψ we have $\psi' \subseteq CL(\psi)$ and $|CL(\psi)| < |\psi|$ where $CL(\psi)$ is the Fischer-Ladner closure of ψ [8], [2], it follows that $|B_\psi| < 2^{|\psi|}$.

The complexity of the transition procedure is thus exponential in the size of the formula ($O(2^{|\psi|})$). However, if ψ is a SCCTL 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 SCCTL we have only state formulae.

Lemma 2. Let $T = \langle S, Lab, Act, \xrightarrow{Act}, s_0\rangle$ be a transition system for a DGP, 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 $|T| = |S| + | \rightarrow |$.

Proof.
The algorithm is based on a product graph of the ABTA By and the transition system T. The size of this product is bounded by $|T| \times |B\psi|$. Like the algorithms proposed in [5], [7], 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 [5]. The run time of the algorithm is thus proportional to the size of the product graph, i.e. $O(|T| \times |B_\psi|)$.

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.

5 Conclusions

The contribution of this paper is the proposition of a new efficient verification algorithm for dialogue game protocols. Our model checking technique allows us to verify the correctness of the protocols in terms of the satisfaction of given properties. 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 SCCTL* 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 the agents'compliance with the semantics of the communicative acts. We also intend to use this technique in order to verify the specifications of multi-agent systems 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 (knowledge, beliefs, intentions, commitments, etc.). Finally, we plan to use this technique to specify and verify agents' trust policies.

Acknowledgments

We would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT) for their financial support. We would also like to thank Josée Desharnais from Laval University (Canada), Yves lespérance from York University (Canada), Frank Dignum from Utrecht University (The Netherlands), and Matteo Baldoni from University of Torino (Italy) for their valuable suggestions and comments.

References

[1] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, *"Verifying protocol conformance for logic-based communicating agents," Computational Logic in Multi-Agent Systems*, vol. 3487 LNAI, pp. 196-212, 2005.

[2] J. Bentahar, *A Pragmatic and Semantic Unified Framework for Agent Communication*. PhD Thesis, Laval University, Canada: 2005.

[3] J. Bentahar, B. Moulin, J-J.Ch. Meyer, and B. Chaib-draa, *"A computational model for conversation policies for agent communication," Computational Logic in Multi-Agent Systems*, vol. 3487 LNAI, pp. 178-195, 2005.

[4] J. Bentahar, B. Moulin, J-J.Ch. Meyer, *"A tableau method for verifying dialogue game protocols for agent communication," Declarative Agent Languages and Technologies*, vol. 3904 LNAI, pp. 223-244, 2006.

[5] G. Bhat, R. Cleaveland, and O. Grumberg, *"Efficient on-the-fly model checking for CTL*," The IEEE Sym-*

posium on Logics in Computer Science, pp. 388-397, 1995.

[6] R.H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge, *"Model checking AgentSpeak," International Conference on AAMAS*, pp. 409-419, 2003.

[7] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis, *"Memory efficient algorithms for verification of temporal properties," Formal Methods in System Design*, vol. 1, pp. 275-288, 1992.

[8] E.A. Emerson, C. Jutla, and A.P. Sistla, *"On model-checking for fragments of m-calculus," Computer Aided Verification*, Courcoubetis, C. (ed.), LNCS 697, pp. 385-396, 1993.

[9] L. Giordano, A. Martelli, and C. Schwind, *"Verifying communicating agents by model checking in a temporal action logic," Logics in Artificial Intelligence*, vol. 3229 LNAI, pp. 57-69, 2004.

[10] M. Kacprzak, A. Lomuscio, and W. Penczek, *"Verification of multiagent systems via unbounded model checking," International Joint Conference on AAMAS*, pp. 638-645, 2004.

[11] P. McBurney, and S. Parsons, *"Games that agents play: A formal framework for dialogues between autonomous agents," Journal of Logic, Language, and Information*, vol. 11 no. 3 pp. 315-334, 2002.

[12] F. Sadri, F. Toni, and P. Torroni, *"Dialogues for negotiation: agent varieties and dialogue sequences," International Workshop on Agents, Theories, Architectures and Languages*, vol. 2333 LNAI, pp. 405-421, 2001.

[13] C. Stirling, and D. Walker, *"Local model checking in the modal Mu-Calculus,"* LNCS 354, pp. 369-383, 1989.

[14] M. Vardi, and P. Wolper, "An automata-theoretic approach to automatic program verification," Symposium on Logic in Computer Science, pp. 332-344, 1986.