Complexity of Multi-Agent Systems Behavior

Michael Dekhtyar [†], **Alexander Dikovsky** [‡], and **Mars Valiev** [§] [¶]

Abstract. The complexity of multi-agent systems behavior properties is studied. The behavior properties are formulated using first order temporal logic languages and are checked relative to the state transition diagram induced by the multi-agent system. Various tight complexity bounds of the behavior properties are established under natural structural and semantic restrictions on agent programs and actions. There are some interesting cases, where the check problem is decidable in deterministic or nondeterministic polynomial time.

Keywords: Intelligent agent, Logic program, Multi-Agent System, Temporal logics, Model checking, Complexity.

1 Introduction

The aim of this paper is to study the verification complexity of behavior properties of intelligent agents systems. Although the intelligent agents are intensively investigated for at least twent

- it is *intelligent,* i.e. its actions are determined by a certain logic and estimation of its environment;

- and it is *goal oriented,* i.e. even if its functioning is continuous, it is oriented on reaching states with some predetermined properties.

Concrete realizations of these properties determine particular agent architectures. Agent's intelligence capacity can vary from finite state control structures or IF-THEN rules to logic programs, non monotone belief based systems or deontic logics (see [20] for a discussion and references).

Basically, the MA-systems we consider in this paper conform to the so called IMPACT architecture introduced and described in detail in the book [20]. This architecture is very elaborated. It includes rather expressive specification means and control structures, e.g. adaptive action bases, logic programs articulating decision policies, constraints, beliefs based meta-reasoning about other agents, uncertainty reasoning, reasoning about time, communication management, security related structures, interfacing, and some other facilities. Such abundance of expressive means makes this architecture well adapted for practical applications. At the same time, it complicates the theoretical investigation of agents' formal properties. The IMPACT architecture agents' semantics is described in terms of transitions between the agents' states and is shown in [20] to be untractable in general. In order to arrive at a polynomial time computable transition semantics, the authors of [20] impose very complex limitations on the agents features. As a result, the definition of such "polynomial" agents becomes bulky.

In this paper, we come up with another and rather simply formulated limitations leading to a polynomial time semantics. We focus on the agent features concerning actions, decision policies and communication. In particular, we do not consider features related to the legacy code, security, metaknowledge structures, temporal and uncertainty reasoning and some others. Moreover, we simplify the internal agent's data structure to be a relational database (in IMPACT a more general structure is allowed), and consider conventional logic programs as a means of an agent action policy definition (IMPACT agent programs include deontic modalities "permitted action", "forbidden action", "action to be performed", etc.). Even after these simplifications, the MA-system architecture remains very rich. So we study the behavior properties under various more or less restrictive constraints on MA-system's parameters and semantics.

We distinguish deterministic and nondeterministic semantics of MA-systems. Their behavior properties are expressed as the properties of trajectories (paths) in the state transition diagrams they induce: a single path in the deterministic case and multiple paths in the nondeterministic case. This allows the use of classical temporal logics: $PLTL$, CTL, CTL^* [9], μ-calculus [16] and their variants as behavior properties description languages. The *"MA-BEHAVIOR"* problem we consider in this paper, consists in verifying that a temporal logic formula Φ holds on the tree of trajectories of a given MA-system. So it is a model checking type problem. Model checking on abstract transition diagrams has been extensively studied since the early 80ies (see [3, 19, 21, 9, 10, 5]). There is however, a substantial difference between the classical problem statement and

that of this paper. Traditionally, the complexity results are established for explicitly presented transition diagrams or else for some of their fixed representations (by a finite automata, by OBDD). We establish the complexity bounds with respect to MA-systems whose operational semantics is presented in the form of transition systems. The novelty of this approach is that the problem complexity is determined by various structural and semantical constraints on MA-systems. MA-systems constitute a compact representation of the corresponding transition system. For example, even for a ground (i.e. variable-free) MA-system A, the transition system $T(A)$ describing its trajectories may have the size exponential in $|A|$, because it can occur that it has $\boldsymbol{O}(2^{|A|})$ states. So sometimes, our lower bounds are more pessimistic as compared with the classical ones for the same classes of logics. As it concerns the upper bounds, either they are more informative and precise (in the case of polynomial time and space complexity), or they are simply translated from the corresponding classical results taking into consideration the size and the number of MA-system states. This being so, we nevertheless establish interesting classes of MA-systems, in which the MA-BEHAVIOR problem turns out to be decidable in deterministic or nondeterministic polynomial time. And this is due to a new possibility of formulating natural constraints in terms of structural parameters of MA-systems.

The paper is organized as follows. In section 2, the architecture of MA-systems is defined and illustrated. In section 3, the temporal logics we use are briefly discussed. In section 4, all results of the paper are presented in the form of a table and briefly commented.

2 Agent and MA-system architecture

We start by presenting our simplified version of the IMPACT architecture.

2.1 Architecture and semantics

A *multi-agent system* (*MA-system*) $\mathcal{A}$ is a finite set $\{a_1, ..., a_n\}$ of *intelligent agents* sharing the same signature. An intelligent agent a, as it shows up in this paper, has its *internal data base (DB)* I_a, which is a finite set of ground atoms. It communicates with the other agents in the system through its message box $MsgBox_a$. The messages that the agents exchange are also ground atoms. The internal DB and the current message box contents make up the agent's current *local state* $IM_a = (I_a, MsgBox_a)$. The set of local states $\{IM_a \mid a \in \mathcal{A}\}$ makes up the *state* of the MA-system.

Each agent is capable of performing a number of parameterized actions constituting its action base AB_a. An action can be seen as an update of the agent's internal DB (defined by a finite list ADD_a of insertions and a finite list DEL_a of deletions of facts) accompanied by a finite list $SEND_a$ of messages to load into the message boxes of recipient agents. Each action α is specified by an *action atom* p_α playing the role of a pattern for the action application. p_α can have variables in common with the action updates and the action messages in the

three lists. For example, in the action
$\alpha = (check\_rel(X,Y),$
$ADD([table1(X,1), table2(X,Y)]),$
$DEL([table1(a_1,Y)]),$
$SEND([(a_1, fax(X,Y,a))]))$
$p_\alpha = check\_rel(X,Y).$

Given a set GA of ground admissible action atoms (defined by the semantics below), each action $\alpha \in AB_a$ uniquely defines the set of admissible ground unifiers σ such that $p_\alpha\sigma \in GA$. As a result, GA uniquely determines the change in the local state $IM_a = (I_a, MsgBox_a)$ resulting from matching actions in AB_a with GA. Namely, the new internal DB $EFF_a(GA, I_a)$ is the result of:

(i) deleting from I_a all instances $\beta\sigma$ of atoms $\beta \in DEL_\alpha$, $\alpha \in AB_a$, such that $p_\alpha\sigma \in GA$,

(ii) and then adding to I_a all instances $\delta\sigma$ of atoms $\delta \in ADD_\alpha$, $\alpha \in AB_a$, such that $p_\alpha\sigma \in GA$.

For example, suppose that matching of the action α above against the current set GA of ground admissible actions defines two admissible unifiers: $\{X \mapsto 0.5, Y \mapsto 3\}$ and $\{X \mapsto 1, Y \mapsto 0\}$. Then the impact of α is that the atoms $table1(a_1,3), table1(a_1,0)$ will be deleted from I_a and then the atoms $table1(0.5,1)$, $table2(0.5,3), table1(1,1), table2(1,0)$ will be added to it.

The parallel effect on message boxes consists in creating for each agent $b \neq a$ the set of messages from a to b, i.e. the set $EFF_a(GA, MsgBox_b)$ of all message instances $\mu\sigma$, where $\mu \in SEND_\alpha$ is addressed to b and $p_\alpha\sigma \in GA$.

In our example, the messages $fax(0.5,3,a), fax(1,0,a)$ will be sent to the agent a_1.

An action α is *expanding* if DEL_α is empty. Agent a is *expanding* if it has only expanding actions.

The policy of behavior of an agent a in its current local state IM_a is determined using its *program* P_a. The intension of P_a in a given local state serves to determine the set of admissible actions GA_a, which itself serves to change this local state through the actions in AB_a as we have explained above.

P_a is a logic program with the clauses of the form $H \leftarrow L_1, ..., L_n$, where $n \geq 0$, the head $H = \alpha(t_1, ..., t_l)$ is the pattern action atom of an action in AB_a, the literals L_i in its body are either action literals, or (extensional) internal DB literals, or atoms of the form $Received(Source\_agent, Message)$, or their negations $\neg Received(Source\_agent, Message)$, or else built-in predicate calls $q(\bar{t})$[1].

An agent's program is *positive* if there are no negations in its clauses. An agent with positive program is also called *positive*.

[1] We adopt a domain closure assumption fixing some finite set of constants **C** denoting the domain objects and considering a set **Π** of polynomial time computable built-in predicates and operations (e.g., the standard arithmetical operations over numbers).

We suppose that the program clauses are *safe* in the sense that all variables in the head H occur *positively* in the body $L_1, ..., L_n$, and that the program $P_a^{state} = P_a \cup \{p \leftarrow \ \mid p \in I_a\} \cup \{Received(Agent\_source, Message) \leftarrow \ \mid (Agent\_source, Message) \in MsgBox_a\}$ is *stratified* [1].

We consider different agent's one-step semantics and respectively different intentional admissible action sets GA_a. All these semantics are defined through the standard minimal model program semantics $Sem(P_a)(I_a, MsgBox_a)$ of P_a in the given local state. $Sem(P_a)(I_a, MsgBox_a)$ is the set M_a^{act} of ground *action atoms* in the minimal model M_a^{state} of P_a^{state}. As it is well known [1], this model is unique for the stratified logic programs and is computed by a polynomial time fixpoint computation procedure from the *groundization* $gr(P_a^{state})$ of the program P_a^{state} [2].

In other words, the semantics of P_a in the given local state is the set of ground actions implied by the program P_a^{state} and available for execution in the agent's current local state.

The agent's one-step semantics should choose (or guess) the set $GA_a \subseteq M_a^{act}$ of intended admissible actions in the set of all available actions M_a^{act}. So it is defined by an operator $Act_a(M)$ applied to a given set of available actions $M = M_a^{act}$. It is natural to suppose that a greater set of available actions leads to a greater set of chosen actions. Therefore, we assume the monotonicity of this operator: $Act_a(M) \subseteq Act_a(M')$ for $M \subseteq M'$. We distinguish *deterministic* and *nondeterministic* one-step semantics.

Deterministic one-step semantics is a function in the class $STEP^D = \{Act : 2^M \rightarrow 2^M \mid Act(M)$ is computable in polynomial time$\}$. For instance, the *total deterministic* semantics defined by $Act^{td}(M) = M$ belongs to this class. This semantics selects the whole M. We can also imagine other types of deterministic one-step semantics, e.g. priority driven deterministic semantics which presumes some partial order $\prec$ on ground actions and is defined by $Act^{\prec d}(M) = \{m \in M \mid \neg\exists m' \in M\ (m' \prec m)\}$. *Deterministic agents* are those having a deterministic one-step semantics in $STEP^D$.

Nondeterministic one-step semantics is a relation Act in the class $STEP^N = \{Act \subseteq 2^M \times 2^M \mid Act$ is recognizable in polynomial time$\}$. The simplest nondeterministic one-step semantics in this class is the *unit choice* one-step semantics defined by $Act^{un}(M) = \{\{p\} \mid p \in M\}$. It guesses some available action in M. Another example is the *spontaneous* one-step semantics defined by $Act^{sn}(M) = \{M' \mid M' \subseteq M\}$. It guesses any subset of available actions in M. *Nondeterministic agents* are those having a nondeterministic one-step semantics in $STEP^N$.

A particular one-step semantics operator Act_a being chosen, it uniquely defines the new local state of a in the way described above.

---

[2] I.e. from the set of all ground instances of clauses in P_a^{state}. It should be noted that the size of $gr(P_a^{state})$ can be exponential with respect to the size of P_a^{state}. We remind that the domain closure assumption we have adopted includes the requirement of polynomial time calculability of the built-in predicates. So the polynomial time complexity of the fixed point computation is preserved.

The *one step semantics* of the MA-system $\mathcal{A}$ induces a one step transition relation $\Rightarrow_{\mathcal{A}}$ on the set $\mathcal{S}_{\mathcal{A}}$ of global states of the form $S =< (I_{a_1}, MsgBox_{a_1}), \ldots, (I_{a_n}, MsgBox_{a_n}) >$. The transition $S \Rightarrow_{\mathcal{A}} S'$ starts by calculating the sets of available actions $M_i^{act} = Sem(P_{a_i})(I_{a_i}, MsgBox_{a_i})$ for all agents $a_i \in \mathcal{A}$. Next, each agent's one-step semantics Act_{a_i} implemented by a deterministic or nondeterministic operator creates *admissible action sets* $GA_{a_i} = Act_{a_i}(M_i^{act})$. The message boxes of all agents in $\mathcal{A}$ are emptied thereafter (so the messages in S are forgotten). Then each agent's internal DB state I_{a_i} is replaced by $EFF_{a_i}(GA_{a_i}, I_{a_i})$ and the message box of each agent a_i is initiated by $\bigcup_{j \neq i} EFF_{a_j}(GA_{a_j}, MsgBox_{a_i})$.

We see that the presented architecture covers the MA-systems of distributed autonomous parallel interacting agents. There are many applications ideally matching this framework, in particular the distributed intelligent programs interacting in local networks. On the other hand, this architecture does not fit asynchronous interactions in the Internet. The nondeterministic semantics we propose are only partially covering such kind of interactions. To find an appropriate concept of the asynchronous MA-system and its verification is an important problem which is out of the scope of this paper.

2.2 Classes of MA-systems

We distinguish two main classes of MA-systems: deterministic and nondeterministic. A MA-system $\mathcal{A}$ is *deterministic* if all its agents are deterministic, otherwise it is *nondeterministic*.

In both classes of MA-systems, we consider the following subclasses induced by natural constraints imposed on agents' components. A MA-system $\mathcal{A} = \{a_1, ..., a_n\}$ is

- *ground* if each program P_{a_i} is ground[3];
- *k-dimensional* if the arities of the action atoms and of the message atoms are bounded by k (*dimension-bounded*, if *k-dimensional* for some k). In fact, this property fixes the maximal number of parameters involved in the actions and in the messages of $\mathcal{A}$;
- *expanding* if all its agents are *expanding*;
- *positive* if all its agents are *positive*;
- *m-agent* if $n \leq m$.
- *r-signal* if there are at most r different ground message atoms (*signals*).

The following simple proposition characterizes the complexity of the MA-system's one step semantics under these restrictions.

Proposition 1
(1) For each deterministic MA-system $\mathcal{A}$, the transition function $S \Rightarrow_{\mathcal{A}} S'$ is computable in polynomial time with respect to $|S| + |\mathcal{A}| + |S'|$ if $\mathcal{A}$ is ground or dimension bounded, and is computable in deterministic exponential time [4] in the

[3] I.e., all its clauses are ground.
[4] In fact, polynomial time with respect to the groundization size.

general nonground case.
(2) For each nondeterministic MA-system $\mathcal{A}$, the transition relation $S \Rightarrow_{\mathcal{A}} S'$ is recognizable in nondeterministic polynomial time with respect to $|S| + |\mathcal{A}| + |S'|$ if $\mathcal{A}$ is ground or dimension bounded, and is recognizable in nondeterministic exponential time in the general nonground case.

2.3 MA-System Behavior

We define the behavior of MA-systems started in an initial global state with empty message boxes. For a MA-system $\mathcal{A}$, its behavior in some initial global state $S^0 =< (I^0_{a_1}, MsgBox^0_{a_1}), ..., (I^0_{a_n}, MsgBox^0_{a_n}) >$, where $MsgBox^0_{a_i} = \emptyset$, $1 \leq i \leq n$, can be seen as the set $\mathcal{T} = \mathcal{T}_{\mathcal{A}}(S_0)$ of infinite trajectories (i.e. sequences of global states) of the form:

$$\tau = (S^0 \Rightarrow_{\mathcal{A}} S^1 \Rightarrow_{\mathcal{A}} ...S^t \Rightarrow_{\mathcal{A}} S^{t+1} \Rightarrow_{\mathcal{A}} ...).$$

For a deterministic MA-system $\mathcal{A}$, $\mathcal{T}$ consists of a single trajectory starting in S^0. If $\mathcal{A}$ is nondeterministic, then $\mathcal{T}$ is an infinite tree of trajectories with the root node S^0. The nodes of $\mathcal{T}$ are the global states $S \in \mathcal{S}_{\mathcal{A}}$ accessible from S^0 by the reflexive-transitive closure of $\Rightarrow_{\mathcal{A}}$. If S is a node of $\mathcal{T}$, then the states in $Next_{\mathcal{A}}(S)$ are its immediate successors in $\mathcal{T}$. An infinite branch of $\mathcal{T}$ starting in some state is a *trajectory* in $\mathcal{T}$.

Let us see two examples of MA-systems. The first system is deterministic.

Example 1 *"Resource-allocation"*
A resource allocation system $\mathcal{RA}$ consists of a manager-agent m distributing among four user-agents u_1, u_2, u_3, u_4 an expendable resource on orders. The user-agents restart their orders periodically and they have different strategies of ordering the resource:
1) u_1 is the first to order a resource; then it repeats its order on receipt of the resource;
2) u_2 orders the next moment after u_1 has ordered;
3) u_3 orders the next moment after u_1 has received the resource from m;
4) u_4 orders every time.
The manager m maintains the list of orders and fulfills the first order on the list, one order at a time. Only one order of each user-agent can be held in the list. So if m receives an order from some user before the previous order of this user has been fulfilled, then the new order is discarded.

We see that the five agents are autonomous in the sense that any of them can function continuously with or without stimuli of the other agents. Meanwhile, these stimuli are necessary in order that the user-agents achieve their goals - to receive the resource. The agents should communicate through messages, which let know that an order has been placed or fulfilled. The intelligence of the four user-agents is rather primitive: just conditional actions. Meanwhile, the agent m must be quite smart in order to control correctly the incoming orders and the states of the queue. This queue and the inventory of messages make up the architecture of the MA-system $\mathcal{RA}$.

The agents of $\mathcal{RA}$ are implemented as follows[5]*. The states I_{u_1} of u_1 can contain the fact put_order. The states I_{u_i} $(i = 2, 3, 4)$ are always empty. The states I_m of m*

[5] Strictly speaking, this MA-system definition does not fit in the constraints above because the program *place_order* is not stratified. However, it can be easily transformed into an equivalent stratified program. We don't do it because the resulting program is larger and less clear.

include the facts of the form $order(X, I)$ *(*$order(u_i, j)$ *means that the order of agent* u_i *is kept in the position* j *in the order list of* m*),* $actual(X)$ *(an order of agent* X *stands on the list of* m*),* $num\_orders(I)$ *(*I *is the number of unfulfilled orders). In order to let* m *and other users know that* u_i *asks for a resource, this agent sends them the message order. When* m *fulfills an order of* u_i*, he sends to* u_i *the message ok.* u_1 *sends to* u_3 *the message ok in order to inform him about the receipt of a resource.*

Agent u_1*. Actions:*

$put: \ ADD = \{put\_order\}, SEND = \{(m, order), (u_2, order)\};$

$receive: \ DEL = \{put\_order\}, SEND = \{(u_3, ok)\};$

$P_{u_1}:$

$put \leftarrow \neg put\_order$

$receive \leftarrow Received(m, ok)$

Agent u_2*. Actions:*

$put: \ SEND = \{(m, order)\};$

$P_{u_2}:$

$put \leftarrow Received(u_1, order)$

Agent u_3*. Actions:*

$put: \ SEND = \{(m, order)\};$

$P_{u_3}:$

$put \leftarrow Received(u_1, ok)$

Agent u_4*. Actions:*

$put: \ SEND = \{(m, order)\};$

$P_{u_4}:$

$put \leftarrow .$

Agent m*. Actions:*

$place\_order(X, I): \ ADD = \{order(X, I), actual(X)\};$

$fulfill\_order(X): \ DEL = \{order(X, 1), actual(X)\}, SEND = \{(X, ok)\};$

$shift(X, I): \ ADD = \{order(X, I)\}, DEL = \{order(X, I + 1)\};$

$new\_num(I, J): ADD = \{num\_orders(J)\}, DEL = \{num\_orders(I)\}$

$P_m:$

$new\_order(X) \leftarrow Received(X, order), \neg actual(X)$

$first\_free(I) \leftarrow num\_orders(I), I > 0$

$first\_free(1) \leftarrow num\_orders(0)$

$place\_order(X, I) \leftarrow new\_order(X), first\_free(I) \quad (X \in \{u_1, u_2\})$

$place\_order(u_3, I) \leftarrow new\_order(u_3), place\_order(X, I - 1) \quad (X \in \{u_1, u_2\})$

$place\_order(u_3, I) \leftarrow new\_order(u_3), first\_free(I), \neg place\_order(u_1, I), \neg place\_order(u_2, I)$

$place\_order(u_4, I) \leftarrow new\_order(u_4), place\_order(u_3, I - 1)$

$place\_order(u_4, I) \leftarrow new\_order(u_4), \neg place\_order(u_3, I - 1), place\_order(X, I - 1) \quad (X \in \{u_1, u_2\})$

$place\_order(u_4, I) \leftarrow new\_order(u_4), first\_free(I), \neg place\_order(u_1, I), \neg place\_order(u_2, I), \neg place\_order(u_3, I - 1)$

$fulfill\_order(X) \leftarrow order(X, 1)$

$shift(X, 1) \leftarrow fulfill\_order(Y), order(X, 2)$

$shift(X, I) \leftarrow shift(Y, I - 1), order(X, I + 1)$

$new\_num(I, J) \leftarrow num\_orders(I), num\_new\_orders(K), J = I + K$

The initial state of m *consists of the fact* $num\_orders(0)$*. Then the fact* $new\_order(X)$ *indicates whether a new order of agent* X *should be placed in the list, the fact* $first\_free(I)$ *defines the position* I *in the list, where a new order should be placed,* $place\_order$ *places new orders at the end of the list in the predefined order* $u_1 < u_2 < u_3 < u_4$*,*

$fulfill\_order$ sends a resource to the first agent in the list, and $shift$ shifts the elements of the list one position to the left, $new\_num(I, J)$ changes the old value of $num\_orders(I)$ by adding the number K of new orders unregistered in $MessageBox_m$ before the step. K is computed by the predicate num_new_orders not defined here.

The second example presents (without implementation details) a nondeterministic MA-system.

Example 2 *"A laboratory Recruiting Committee (RC) "*
The RC consists of six deterministic agents: agents-members $m_i, 0 \leq i \leq 4$ (m_0 being the laboratory director), and agent-secretary s, and one spontaneous nondeterministic agent c simulating at each recruitment cycle the submission of a number of candidatures $cand(C, Profile, Merits, Grants)$. C stands for the candidate's identification. $Profile$ stands for one of the activity profiles: lp (logic programming), ai (artificial intelligence), cl (computational linguistics) and mm (multimedia). An integer $Merits$ evaluates the candidate's scientific merits. An integer $Grants$ represents the sum of the grants the candidate has obtained. The only action of c is to introduce the set of all possible candidatures. Due to a spontaneous choice, a particular set of candidatures submitted for the new recruitment cycle is created. Besides the candidatures, the information about the staff member selected in the preceding recruitment is also available: $selected(C, Profile)$. The secretary s controls the vote through messages. Before the vote, the members may speak out on the candidates (through messages). Then they vote by sending their secrete vote messages to the secretary: the selected candidate or the abstention. On receiving all members' votes, the secretary updates the information on the selected staff member according to the score and deletes the nonselected candidatures.

The RC members have different vote tactics but all of them abstain when their criteria do not ensure the uniqueness of choice:
m_0 *votes according to the following profile preferences: $lp > ai > mm > cl$, selects a candidate with the best grants sum and announces his choice to all RC members;*
m_1 *always votes for the candidate of m_0;*
m_2 *selects a candidate with a profile different from that chosen by m_0 and having the best grants sum;*
m_3 *votes for the candidate with the best scientific merits whose profile is different from that of the last recruited candidate;*
m_4 *votes with the majority, if any; if there is no majority, then m_4 chooses the "anti-boss" party candidate.*

3 Verification of behavior properties

The crucial point about the examples above is that the behavior of the MA-systems $\mathcal{RA}$ and $\mathcal{RC}$ should verify some properties, e.g. the behavior of $\mathcal{RA}$ in example 1 should be *fair* in the sense that each user-agent is repeatedly served by m (i.e. served sometimes in the future after its order has been fulfilled). This property should be verified with respect to all runs of $\mathcal{RA}$.

We represent the MA-systems as transition systems on global states. In these terms, a run of a deterministic MA-system corresponds to a trajectory through states, each two adjacent states in the trajectory being related by a transition. As for nondeterministic MA-systems, a run of such a system corresponds to a trajectory in the tree of trajectories with the same condition on the adjacent states.

This makes the logics of linear or branching time a convenient tool of behavior specification and analysis. In particular, linear time logics are well suited for the deterministic MA-system behavior description, whereas the branching time logics and more expressive μ-calculus are more adapted to the nondeterministic MA-system behavior properties.

The *"MA-BEHAVIOR"* problem we consider in this paper, applies to deterministic MA-systems as well as to nondeterministic ones. Given such a system $\mathcal{A}$, an initial global state S_0 and a formula Φ expressing a property of trajectories, the MA-BEHAVIOR problem $\mathcal{A}, S_0, \Phi$ has a positive solution if Φ holds on the tree $\mathcal{T}_{\mathcal{A}}(S_0)$ of trajectories of $\mathcal{A}$ starting in S_0 (denoted $\mathcal{T}_{\mathcal{A}}(S_0), S_0 \models \Phi$). We see that it is a kind of the model checking problem, though applied to MA-systems in the role of transition diagram specification.

In order to specify the behavior properties of the deterministic MA-systems, we use the *propositional linear temporal logic* (PLTL) and its first order extension FLTL [9] with the standard linear time operators **X** ("nexttime"), **U** ("until"), **V** ("unless"), **G** ("always"), and **F** ("sometimes"). For nondeterministic MA-systems, we use simple subsets of classical branching time logics: $\forall$LTL and $\exists$LTL consisting respectively of the formulas of the form $\mathbf{E}(\phi), \mathbf{A}(\phi)$, where ϕ is in $FLTL$ [6], first order extension of the μ-calculus [16, 10] (FO-μ) and its alternation depth bounded subsets (FO-$\mu_l, l \geq 1$).

For example, the MA-system $\mathcal{RA}$ above being deterministic, its fairness is expressed by the following simple LTL formula : $\bigwedge_{i=1}^{4}$ **G F** $receives_{u_i}$ (for each of the four user-agents u_i and at any time point, it is true that it will receive the resource in the future). As for the MA-system $\mathcal{RC}$ in example 2, we prove elsewhere that its behavior can have anomalies, e.g. the worst candidate can be selected under certain conditions.

4 Main results

The main results of the paper are summarized in the table below [7].

The first four columns in this table categorize the classes of the MA-systems we consider. In the column **Type** the deterministic and nondeterministic MA-systems are distinguished. The columns **Gr** and **Exp** serve to distinguish between ground/non-ground and expanding/non-expanding systems. In the column **Parameters**, several architectural parameters are used to impose constraints leading to low complexity classes. In particular, N denotes the size of the system, m is the number of agents, k is the maximal arity of the action and message atoms (in the propositional case $k = 0$), r is the number of signals (i.e. of messages). The fifth column **Logic** serves to indicate the particular temporal logic used as a property formulation language. The sixth column presents the results obtained in the paper and the seventh column contains the references to

[6] The path quantifiers **A** and **E** mean respectively "for all trajectories" and "for some trajectory".

[7] The results are formulated without proofs due to space limitations.

these results. Each row of the sixth column indicates the complexity class of the problem identified by the corresponding rows of the first five columns.

Type	**Gr**	**Exp**	**Parameters**	**Logic**	**Complexity**	**Nmb**
Deter-minis-tic	Yes	Yes	positive	PLTL	P	d1
			$m^2 * r = O(\log N)$	PLTL	P	d2
			fixed $m \geq 2$	PLTL	PSPACE	d3
			fixed $r \geq 1$	PLTL	PSPACE	d4
		No		FLTL	PSPACE	d5
	No	Yes	positive, fixed k	PLTL	P	d6
			positive	PLTL	EXPTIME	d7
		No	fixed k	FLTL	PSPACE	d8
				FLTL	EXPSPACE	d9
Non-deter-minis-tic	Yes	Yes	$m^2 * r = O(\log N)$	$\exists$ LTL	NP	n1
				$\forall$ LTL	co-NP	n2
			fixed $m \geq 2$	FO-μ_1	EXPTIME	n3
			fixed $r \geq 1$	FO-μ_1	EXPTIME	n4
		No		FO-μ_l, fixed l	EXPTIME	n5
				FO-μ	in NEXPTIME $\cap$ co-NEXPTIME	n6
	No	Yes	fixed $r \geq 1$	$\exists$ LTL	NEXPTIME	n7
				$\forall$ LTL	co-NEXPTIME	n8
			positive	$\exists$ LTL	EXPSPACE-hard	n9
		No	fixed k	FO-μ	EXPTIME	n10
				FO-μ_l, fixed l	EXPEXPTIME	n11
				FO-μ	in NEXPEXPTIME $\cap$ co-NEXPEXPTIME	n12

In all rows except n6, n9 and n12, we mean that the corresponding problem is complete in the indicated complexity class. In the rows n6 and n12 only upper complexity bounds are indicated, whereas in the row n9, only lower bound is stated. It is not surprising that in the general case the MA-BEHAVIOR problem is very complex: EXPSPACE-complete for the deterministic systems (d9), and EXPEXPTIME-complete for nondeterministic systems (n11). However, it turns out to be tractable for the deterministic systems under some restrictions reasonable for practical applications. Namely, it is polynomial time decidable under strong monotonicity conditions (d1, d6), or else under the condition that the system is expanding with the additional structural restriction $m^2 * r = O(\log N)$ (d2). The theorems (d3, d4, d7) show that removing any of these constraints leads to intractability. In fact, the theorem (d7) holds even for 0-signal 1-agent systems. The origin of the restriction $m^2 * r = O(\log N)$ is that the trajectory of a deterministic system $\mathcal{A}$ is periodic with some pre-period q and period p, and that under this restriction, $q + p \leq mN2^{rm^2} \leq cmN^2$ for some constant c, when $\mathcal{A}$ is ground and expanding. In particular, the theorems (d3, d4) show that ground expanding MA-systems cannot be simulated in polynomial time by

single ground expanding agents (facts simulating messages should be updated), so the distribution really plays a role.

As for the nondeterministic systems, it would be difficult to hope that the behavior verification problem were polynomial time decidable. It seems likely that the theorems (n1,n2) stating (co-) nondeterministic polynomial time complexity of the problem in the case of nondeterministic MA-systems give the best results one might expect in this case. As in deterministic case, the theorems (n3,n4,n5) show that elimination of any of the constraints on r, m or on the deletions leads to exponential time complexity. In fact, in all these cases a more exact lower bound of the form $\boldsymbol{O}(c^{n/log\ n})$ can be obtained. It should be noted that all upper bounds concerning FO-μ, FO-μ_1, FO-μ_l are simple generalizations of the well known result in [10] on model checking complexity for the propositional μ-calculus.

5 Conclusion

The agent and MA-system architectures published within the past few years are dissimilar and diversified because they represent various application domains of this new software technology. Our study concerns one such specific architecture which covers the systems of distributed autonomous parallel interacting agents. On the other hand, this architecture does not fit exactly the asynchronous distributed algorithms frame. The nondeterministic semantics we propose are only partially covering such kind of interactions. To find an appropriate concept of the asynchronous MA-system and its verification is an important problem, which is out of the scope of this paper.

This paper illustrates the way in which penetrating in much detail in a complex MA-system architecture permits in some cases to better understand the behavior properties and in this way, to obtain more deep results. Considered from this point of view, this paper creates a perspective of applying similar analysis to other MA-system architectures in order to find interesting subclasses of MA-systems with efficiently checked behavior properties.

References

1. Apt, K. R., Logic Programming. In: J. van Leeuwen (Ed.) *Handbook of Theoretical Computer Science. Volume B. Formal Models and Semantics, Chapter 10*, Elsevier Science Publishers B.V. 1990, 493-574.
2. Bylander, T., The computational Complexity of Propositional STRIPS Planning, *Artificial Intelligence,* 69:165–204, 1994.
3. Clarke, E. M., Emerson, E. A. Design and synthesis of synchronization skeletons using branching time temporal logic. In: *Proc.of Workshop on Logics of Programs, Lecture Notes in Computer Science,* N. 181, 1981, 52-71.
4. Calvanese, D., De Giacomo, G., and Vardi, M.Y., Reasoning about actions and planning in LTL action theories, *Proc. of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'02),* 593-602, 2002.
5. Clarke, E. M., Grumberg, O. and Peled, D., *Model Checking,* MIT Press, 2000.

6. Dekhtyar, M., Dikovsky, A., On Homeostatic Behavior of Dynamic Deductive Data Bases. In: *Proc. 2nd Int. A.P.Ershov Memorial Conference "Perspective of Systems Informatics", Lect. Notes in CS,* N. 1181, 1996, 420-432.
7. Dekhtyar, M., Dikovsky, A., and Valiev, M., Applying temporal logic to analysis of behavior of cooperating logic programs. *Lect. Notes in CS,* N. 1755, 2000, 228-234.
8. Eiter, T., Fink,M., Sabbatini, G., and Tompits, H., "Reasoning about Evolving Nonmonotonic Knowledge Bases", *Proc. of LPAR01* , 2001.
9. Emerson, E. A. Temporal and modal logic. In: J. van Leeuwen (Ed.), *"Handbook of Theor. Comput. Sci.",* Elsevier Sci. Publishers, 1990.
10. Emerson, E. A. Model checking and the mu-calculus. In: N. Immerman, P. H. Kolaitis (Eds.), "Descriptive Complexity and Finite Models". *Proc. of a DIMACS Workshop,* 1996, 185-214.
11. Erol, K., Nau, D.S., and Subrahmanian, V.S., Complexity, Decidability and Undecidability Results for Domain-Independent Planning, *Artificial Intelligence Journal,* 76(1-2):75-88, 1995.
12. Fagin, R., Halpern, J.Y., Moses, Y. and Vardi, M., *Reasoning about Knowledge,* 1995, MIT Press.
13. Fisher, M., Wooldridge, M., Specifying and Verifying Distributed Intelligent Systems. In: M. Filgueiras and L. Damas (Eds.) *Progress in Artificial Intelligence – Sixth Portuguese Conf. on Artificial Intelligence. LNAI,* N. 727, 1993, pp.13-28, Springer-Verlag: Berlin, Germany.
14. Jennings, N., Sycara, K. and Wooldridge, M. A roadmap of agent research and development *Autonomous Agents and Multi-Agent Systems,* 1998, 1(1):7-38.
15. Müller, J. P., Architectures and applications of intelligent agents: A survey. *The Knowledge Engineering Review,* 1998, 13(4):353-380.
16. Kozen, D., Results on the Propositional μ-calculus. *Theoretical Computer Science*, 1983, v. 27, pp. 333–354.
17. Petrie, C., What is an agent? In: J.P. Müller, M. J. Wooldridge, and N. R. Jennings (Eds.) *Intelligent Agents III – Proc. of the Third Intern. Workshop on Agent Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence,* N. 1193, 41-43. Springer-Verlag.
18. Reiter, R. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.
19. Sistla, A. P., Clarke, E. M., The complexity of propositional linear temporal logic. J.ACM, 32(3), 1985, 733-749.
20. Subrahmanian, V. S., Bonatti, P., Dix, J., et al., *Heterogeneous Agent Systems,* MIT Press, 2000.
21. Vardi, M., Wolper, P., An automata-theoretic approach to automatic program verification. In: *Proc. of the IEEE Symposium on Logic in Computer Science,* 1986, 332-344.
22. Wooldridge, M., The Computational Complexity of Agent Design Problem. In: E. Durfee, (Ed.) *Proc. of the Fourth Intern. Conf. on Multi-Agent Systems (ICMAS 2000),* IEEE Press, 2000.
23. Wooldridge, M., Jennings N. Intelligent agents: Theory and practice. *The Knowledge Engineering Review,* 1995, 10(2).