Applications of model checking for multi-agent systems: verification of diagnosability and recoverability

Franco Raimondi
Department of Computer Science
University College London
London, UK

Charles Pecheur
Department of Computer Science and Engineering
Université catholique de Louvain
Louvain La Neuve, Belgium

Alessio Lomuscio
Department of Computer Science
University College London
London, UK

Abstract

This paper presents a practical application of model checking for multi-agent systems to the automatic verification of diagnosability. First, a characterisation of diagnosability in terms of epistemic properties of agents is given; then, experimental results are presented for preliminary investigations in the automatic verification of diagnosability of Livingstone models.

1 Introduction

The last fifteen years have seen an increasing interest in the use of *automatic* techniques for the verification of complex systems, both for hardware and software components. The introduction of *symbolic* techniques (as in [6]) has enabled the development of various model checkers, including SMV [20], SPIN [15], NuSMV [7], and Verics [21]. Traditionally, these techniques and tools have been developed for *temporal* logics only, i.e. Computational Tree Logic (CTL) in the case of SMV and NuSMV, and Linear Temporal Logic (LTL) for SPIN.

Recently, various extensions of model checking techniques and tools have been investigated for the verification of richer modal logics that include modal operators to reason about time, knowledge, beliefs, and strategies. For instance, [2] introduces the logic ATL, an extension of CTL to reason about strategies, and [1] presents MOCHA, a symbolic model checker for ATL. The logic ATEL, an extension of ATL with epistemic operators for multi-agent system, is presented in [14]. In [3] the authors reason about the verification of various aspects of agents, and [4] presents a methodology for reducing the problem of model checking agents to the problem of verifying temporal models. Symbolic techniques for temporal-only logics have also been extended to richer logics: the model checker Verics [21] uses SAT-based techniques for the verification of temporal and epistemic properties of agents. The model checker MCK [12] is based on ordered binary decision diagrams (OBDDs), and supports the verification of epistemic and temporal properties for systems defined on interpreted systems semantics. Similarly, the tool MCMAS [23] uses OBDDs and allows to reason about time, knowledge, and correct behaviour of agents. The development of these tools is motivated by the interest in the automatic verification of various scenarios in which it seems more natural to reason about epistemic properties of multi-agent systems [28], as in communication and security protocols.

The aim of this paper is to present another application of model checking for multi-agent systems. In particular, we investigate how *diagnosability* and *recoverability* can be expressed as temporal-epistemic *specification patterns*. Diagnosability and recoverability properties of a system correspond, respectively, to the feasibility of diagnosing and recovering from faults in that system, given available observable and controllable variables (sensors and actuators). In this sense, they are particular forms of observability/controllability properties found in classical control theory. Diagnosability has been studied by several authors in the domain of discrete systems [25, 26, 17, 16, 10, 8] and timed systems [27]. In particular, [8], shows how diagnosability properties can be transformed into reachability properties on a derived model, and be verified using BDD- and SAT-based symbolic model checking in the NuSMV tool [7].

These approaches differ from the one presented here in that we represent diagnosability by using epistemic and temporal properties of *agents*. Following this, we use a model checker for multi-agent systems (MCMAS [23]) to verify an example from Livingstone, “a model-based health monitoring system developed at NASA Ames Research Center” [22]. In this example we test diagnosability and recoverability properties expressed as temporal-epistemic properties of agents. Further, we compare our results with the ones obtained using NuSMV, a temporal-only model checker.

The rest of the paper is organised as follows. In Section 2 we review the model checker MCMAS and its underlying semantics (deontic interpreted systems), the Livingstone system, and the concept of diagnosability in temporal models. In Section 3 we characterise diagnosability and recoverability in terms of epistemic operators; we present experimental results of our approach for an example from the Livingstone suite in Section 4. We discuss the results and conclude in Section 5.

```
Agent SampleAgent
 Lstate = {s0,s1,s2,s3};
 Lgreen = {s0,s1,s2};
 Action = {a1,a2,a3};
 Protocol:
   s0: {a1};
   s1: {a2};
   s2: {a1,a3};
   s3: {a2,a3};
 end Protocol
 Ev:
   s2 if ((AnotherAgent.Action=a7);
   s3 if Lstate=s2;
 end Ev
end Agent
```

Figure 1: An ISPL program

2 Preliminaries

2.1 MCMAS and deontic interpreted systems

MCMAS [23, 24] is a model checker for the verification of temporal, epistemic, and correctness properties of agents. MCMAS extends to more complex logics the traditional model checking algorithms for CTL [9] and uses OBDDs [5] as an efficient encoding technique. Agents are specified by using the ISPL language (Interpreted Systems Programming Language). An example of an ISPL program is given in Figure 1. We refer to [23] for the full syntax of ISPL.

An ISPL program is the description of a *deontic interpreted system*. A deontic interpreted system [11, 19] is a formal description of a system of n agents. Each agent i in the system is characterised by a private set of *local states* L_i and by a set of *actions* Act_i, that are performed publicly. The set L_i is partitioned into two disjoint sets: a non-empty set G_i of "green" (or correct) states and a set R_i of "red" (or faulty) states. Actions are performed in compliance with a given protocol: a protocol $P_i : L_i \rightarrow 2^{Act_i}$ for agent i is a function from local states of i to actions of i. Each agent evolves by modifying its local state, as prescribed by an evolution function $t_i : L_i \times L_E \times Act \rightarrow L_i$. In the previous definition, $Act = Act_1 \times \ldots \times Act_n \times Act_E$ denotes the set of "joint" actions, and L_E and Act_E are, respectively, the local states and the actions of a special agent, the *environment*, which is used to model the environment in which agents operate. Figure 1 describes an agent characterised by four local states (`s0, s1, s2, s3`), three of which are "green"; the agent in the figure is allowed to perform three distinct actions (`a1, a2, a3`), in compliance with the protocol. The agent changes its local state to either `s2` or `s3` if the relevant condition on the right hand side of the expression in the description of the evolution function is true, and does not change its local state if none of the conditions is true. Agents evolve synchronously: at each time step, all agents satisfying some transition condition change their local state at the same time.

An element g of the Cartesian product of the local states of the agents and the environment is called a *global state*. We will denote with $S = L_1 \times \ldots \times L_n \times L_E$ the set of all possible global states. Given a set of initial states $I \subseteq S$, the protocols and the evolution functions generate the set of *reachable global state*, denoted with G.

Given a set of atomic propositions AP, the labelling function $h : AP \to 2^S$ returns the set of global states in which a proposition is true. Formally, given a set of agents $\Sigma = \{1, \ldots, n\}$, we will denote a deontic interpreted system by the following tuple:

$$DIS = \left\langle (G_i, R_i, Act_i, P_i, t_i)_{i \in \Sigma}, (G_E, R_E, Act_E, P_E, t_E), I, h \right\rangle.$$

Logic formulae to reason about time, knowledge, and correct behaviour of agents can be evaluated on deontic interpreted systems. Consider the following language:

$$\varphi \quad ::= \quad p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E(\varphi U \psi) \mid \ K_i\varphi \mid E_\Gamma\varphi \mid C_\Gamma\varphi \mid D_\Gamma\varphi \mid \ \widehat{K}_i^j\,\varphi.$$

In the syntax above, $p \in AP$ is an atomic proposition, EX, EG, and EU are standard branching time operators [9], $K_i\varphi$ expresses that agent i knows φ, $\Gamma \subseteq \Sigma$ is a set of agents, $E_\Gamma\varphi, C_\Gamma\varphi$, and $D_\Gamma\varphi$ express, respectively, that everybody in a group knows φ, that φ is common knowledge in a group, and that φ is distributed knowledge in a group (we refer to [11] for more details). The operator $\widehat{K}_i^j$ [19] is used to express the knowledge of agent i under the assumption that agent j is functioning correctly.

To define the semantics of well-formed formulae, a standard Kripke model $M_{DIS} = (W, R_t, \sim_1, \ldots, \sim_n, R_1^O, \ldots, R_n^O, h)$ is associated to DIS, as follows:

- The set W of possible worlds is G (the set of reachable states of DIS).
- $R_t \subseteq W \times W$ is a (serial) temporal relation. Two worlds w and w' are such that $R_t(w, w')$ iff there exists $a \in Act$ such that $t(w, a) = w'$. The function t is a "global" evolution function for DIS, defined as: $t(g, a) = g'$ iff for all $i \in \Sigma, t_i(l_i(g), a) = l_i(g')$, and $t_E(l_E(g), a) = l_E(g')$ (where $l_i(g)$ denotes the i-th component of global state g, corresponding to the local state of agent i).
- $\sim_i$ are epistemic relations, defined by considering the equality of local states. Two worlds w and w' are such that $\sim_i (w, w')$ iff $l_i(w) = l_i(w')$ [11].
- R_i^O are relations for correctness. Two worlds w and w' are such that $R_i^O(w, w')$ iff $l_i(w') \in G_i$ [19].
- The labelling function h is defined as above.

Satisfiability is defined in M_{DIS} in the standard way. We refer to [24] for more details, and we report only the satisfaction conditions for K_i, C_Γ, D_Γ, and $\widehat{K}_i^j$:

$M_{DIS}, w \models K_i\varphi$	iff	for all $w' \in W$, $\sim_i (w, w')$ implies $M_{DIS}, w' \models \varphi$,
$M_{DIS}, w \models C_\Gamma\varphi$	iff	for all $w' \in W$, $[R_\Gamma^E]^*(w, w')$ implies $M_{DIS}, w' \models \varphi$, where $[R_\Gamma^E]^*$ is the transitive closure of the relation $R_\Gamma^E = \bigcup_{i \in \Gamma} \sim_i$,
$M_{DIS}, w \models D_\Gamma\varphi$	iff	for all $w' \in W$, $R_\Gamma^D(w, w')$ implies $M_{DIS}, w' \models \varphi$, where $R_\Gamma^D = \bigcap_{i \in \Gamma} \sim_i$,
$M_{DIS}, w \models \widehat{K}_i^j\,\varphi$	iff	for all $w' \in W$, $\sim_i (w, w')$ and $R_j^O(w, w')$ implies $M_{DIS}, w' \models \varphi$,

MCMAS takes as an input a description of a DIS using ISPL and builds a symbolic representation of M_{DIS}. Formulae are provided to MCMAS via a text file, and verification is performed by using algorithms based on OBDDs by computing fix-points and comparing OBDDs. Using this technique various scenarios for communication and security protocols have been tested [24].

2.2 Livingstone and diagnosability

Livingstone is "a model-based health monitoring system developed at NASA Ames Research Center" [22]. Livingstone is responsible for the diagnosis and recovery of systems. To this end, Livingstone "observes" the commands given to a system and "reads" the outputs, to build the best possible "picture" of a particular system at a given time.

Livingstone has been used for various modelling scenarios, including a propellant production plant for Mars, the fuel feeding subsystem of a next-generation space shuttle, and as a part of an AI-software called Remote Agent (RA), which is "a software designed to operate space crafts with minimal human assistance" [22] (RA flew the experimental space craft DS1 between the 17th of May and the 21st of May 1999).

Models of Livingstone are described in a Java-like language called JMPL. *Formal verification* of Livingstone models can be performed via model checking. A methodology to translate Livingstone models into SMV models is described in [22]. Also, a software tool called `jmpl2smv` is provided to perform the translation automatically.

Due to its specific tasks, one of the most important Livingstone's properties that may be verified is the possibility of performing a correct *diagnosis* of a system. A formal framework for *diagnosability* is defined in [8]; a *diagnosis condition* is defined as a pair $c_1 \perp c_2$ of non-empty sets of states c_1 and c_2 of the system, where $\perp$ is a separator for the two set of states meaning that the pair is a diagnosis condition. Intuitively, $(c_1 \perp c_2)$ is *diagnosable* iff there are no two execution traces π_1 and π_2 from the initial state such that all the observables (commands and outputs) are equal along the traces, and π_1 leads to c_1 and π_2 to c_2. For instance, fault detection can be expressed in terms of a diagnosis condition as $(fault \perp \neg fault)$. This means that there are no two traces such that one trace leads to a faulty state, and the other to a non-faulty state. We refer to [8] for more details.

A temporal-only model checker, like NuSMV, can be used for the formal verification of diagnosability [8]. The key idea is to build two copies of the system, and to constrain the observable variables of both copies to be equal using the tools provided by the model checker (e.g., using the `INVAR` construction in NuSMV). Effectively, a copy of the system may be denoted with `test` and the other copy with `twin`. In this case, fault detection can be expressed by the CTL formula $AG(\neg(\texttt{test.mode} = faulty \wedge \texttt{twin.mode} \neq faulty))$. To allow for the verification of diagnosability, the translator from JMPL to SMV has been extended with an option to output an SMV file with "twin" modules. Using this technique, *diagnosability* has been verified in a number of examples (see [8] for more details).

3 Diagnosability as an epistemic specification

In many specification examples, instead of using temporal properties only, it is at times easier to reason about higher order properties such as knowledge. For instance, *diagnosability* can be naturally expressed by ascribing a form of knowledge to a diagnoser: a diagnoser is able to diagnose a condition $(c_1 \perp c_2)$ iff the diagnoser always knows whether $\neg c_1$ or $\neg c_2$.

This last sentence can be expressed formally in the framework of deontic interpreted systems and can be automatically verified in MCMAS. The system in which diagnosability needs to be verified may be modelled by means of a deontic interpreted system, in which the diagnoser D is a particular agent that stores in its local states the outputs and the commands; the original system may be modelled by another agent, "observed" by the diagnoser. Thus, diagnosability can be expressed by the formula $AG(K_D(\neg c_1) \vee K_D(\neg c_2))$, where the knowledge operator forces the observable variables to remain unchanged. Notice that this definition is equivalent to the definition of diagnosis provided in Section 2.2 but, instead of relying on "twin" models, it is based solely on one model, and analysed in terms of the epistemic properties of the diagnoser.

Deontic interpreted systems allow also to reason about epistemic properties of a group of agents. In particular, *distributed knowledge* in a group Γ is the knowledge that the group has if every agent in Γ "shares" his knowledge with the other agents of Γ. This allows for a generalisation of the concept of diagnosability. Let $\Delta \subseteq \Sigma$ be a subset of the set of agents; intuitively, Δ is a set of diagnosers, and each diagnoser in Δ is responsible for the monitoring of a particular aspect of the system (i.e. a part of the outputs, or a part of the commands), while ignoring the remainder. We say that a condition $(c_1 \perp c_2)$ is diagnosable by group Δ iff $AG(D_\Delta(\neg c_1) \vee D_\Delta(\neg c_2))$.

Correctness can also be introduced when reasoning about knowledge: the operator $\widehat{K}_i^j$ expresses what agent i knows, under the assumption that agent j is working correctly. The concept of knowledge under the assumption of correct behaviour can be extended easily to the concept of distributed knowledge in a group under the assumption of correct behaviour of another group. We will use the operator $\widehat{D}_\Delta^\Gamma$ to denote the distributed knowledge of group Δ, assuming that agents in Γ are working correctly. In this way, it is possible to reason about *diagnosability under the assumption of correct behaviour*. We will provide an example of its application in Section 4.

3.1 Verification of recoverability

Recoverability is the ability of a system to recover from some "faulty" state. The aim of this section is to provide a formal description of properties such as "the diagnoser knows that, assuming that the server will operate in normal conditions, the server will recover from its current faulty state".

Instead of using the accessibility relations R_i^O, correct behaviour can also be characterised in terms of *local propositions* [19] (local propositions have been used in a similar way in [13] for the verification of epistemic properties using a temporal-only model checker). Let $g_i \in AP$ be a proposition which is true only in the green states of agent i. Then, for any deontic interpreted system DIS, the following equivalence holds:

$$M_{DIS} \models \widehat{K}_i^j \varphi \Leftrightarrow K_i(g_j \rightarrow \varphi).$$

Let $f \in AP$ be a proposition denoting some faulty state, and let φ be a formula denoting some desired states of affairs. Then, the ability of a diagnoser Δ to diagnose recoverability from f on a deontic interpreted system DIS, assuming "normal" operating conditions for the agents in Γ, can be expressed as

$$M_{DIS} \models f \rightarrow D_{\Delta}(\neg E[\Gamma_i \; U \; (\Gamma_i \wedge \neg\varphi)]).$$

In the previous expression, $\Gamma_i = \bigwedge_{i \in \Gamma} g_i$ is a boolean expression composed by the conjunction of the local proposition denoting correct behaviour for agents in Γ. Intuitively, the "until" part of the formula states that there is no "correct" temporal path from "faulty" states which will not reach a state in which φ holds. This fact, in turn, is distributed knowledge between the diagnosers.

Section 4 provides an example for the verification of diagnosability of recoverability.

4 Diagnosability with MCMAS

Two different definitions of diagnosability have been provided in Section 2.2 and Section 3: the former using "twin" models, and the latter using an epistemic characterisation. In the Livingstone framework, a "twin model" can be obtained automatically by using the `jmpl2smv` tool, and the twin model can be verified by using the model checker NuSMV.

To compare this approach with the verification of diagnosability as an epistemic property, we consider the translation of a Livingstone model into a deontic interpreted system expressed in the syntax of the tool MCMAS. We analyse a test example from Livingstone: a circuit composed by a cascade of circuit breakers, a source, and LEDs. The circuit is represented in Figure 2.

Each circuit breaker is allowed to be in one of the following states: `on, off, tripped, blown, ufault`. `on` and `off` are "green" states. `tripped` is a resettable fault, `blown` is a non recoverable fault, and `ufault` denotes an unknown fault. A Controller sends (arbitrary) commands to the circuit breakers, and a Diagnoser reads the commands and the outputs as defined in the Livingstone model.

Various assumptions can be made while modelling this example as a deontic interpreted system DIS_1. Here we will consider the following (other variations on this example are available from [23]):

- Each circuit breaker is an agent; each led is an agent; the source is an agent.
- For each circuit breaker, we assume the existence of a commander agent allowed to send random commands to the circuit breakers.
- We assume the existence of two diagnosers: the first can see the output of the source, the second can see the LEDs.

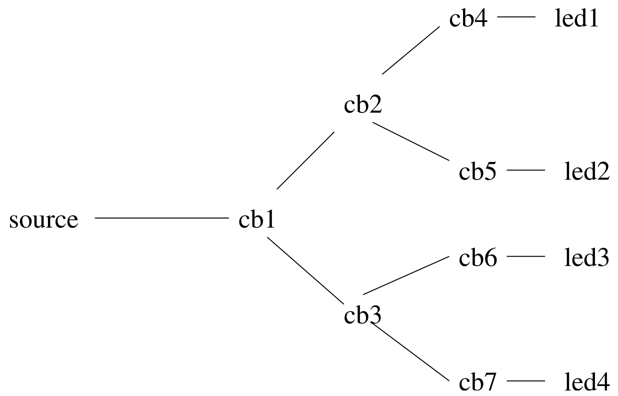

Figure 2: Livingstone example: a circuit

Tool	**Time**	**OBDDs vars**
MCMAS	2.39sec	90
NuSMV	10.47sec	235

Table 1: Average verification results

Due to space limitations we refer to the files available online for the full ISPL code of the example.

As an example of diagnosability, we check that the diagnoser obtained by considering the distributed knowledge of the two diagnosers is not able to detect faults correctly, i.e. the formula $AG(D_\Delta(\textit{faulty}) \vee K_D(\neg \textit{faulty}))$ is false (here $\textit{faulty} = \bigvee f_i$ is a proposition denoting that some component i of the circuit is not working correctly and Δ is the group composed by the two diagnoser agents for the output and the LEDs). This is because, under our assumptions, the diagnoser is not able to distinguish between "correct" `off` states and faulty states. The same result can be obtained by using the twin model of the circuit and by verifying it using NuSMV. The experimental results for this example and this formula are reported in Table 1.

As an example of recoverability we can check that the following is not true under our assumptions:

$$M_{DIS_1} \not\models \textit{faulty} \rightarrow D_\Delta(\neg E[\mathbf{g\_cb}\; U\; (\mathbf{g\_cb} \wedge \neg \mathbf{led\_on})]).$$

In the previous expression $\mathbf{g\_cb}$ is a formula true in the green states of the circuit breakers, and it is obtained by considering the conjunction of the local propositions expressing correct behaviour for all the circuit breakers. The proposition $\mathbf{led\_on}$ denotes a state in which LEDs are on. Intuitively, the diagnoser is not able to diagnose recoverability because circuit breakers may be in an unrecoverable faulty state, and the diagnoser cannot distinguish it from recoverable states.

Verification of this last formula took the same amount of time and memory than the previous example.

5 Discussion and conclusion

Table 1 shows that the use of the epistemic characterisation of diagnosability may reduce the number of boolean variables needed for encoding the system symbolically. Also, thanks to the more compact representation, verification is faster with MCMAS. However, the comparison of Table 1 does not take into account various differences between MCMAS and NuSMV. The most relevant difference is that diagnosers in MCMAS do not have *perfect recall* [11], i.e. they cannot "remember" past observations; on the contrary, diagnosability using "twin models" is verified under the assumption of perfect recall. It is still possible to consider a form of "bounded" recall for agents by modifying the structure of local states in MCMAS, but we did not investigate this issue further. When "perfect recall" is needed, MCK [12] may provide a better verification tool for Livingstone models because it implements perfect recall natively, but for a limited class of formulae.

Also, NuSMV implements features that are not present in MCMAS, such as fairness conditions, invariants (`INVAR`), generation of counter-examples, and verification via bounded model checking. While limited only to a certain class of formulae, bounded model checking has been proven significantly more effective for the verification of diagnosability than OBDD-based techniques [8]. In this line, it may be worth investigating the feasibility of using Verics [18] in the verification of Livingstone models.

Another disadvantage of MCMAS is that its ISPL input language is not well suited for the description of hardware components, such as the circuit of the example above. The description of a system using ISPL is based on agents, and each agent has only a (single) set of local states; moreover, agents may perform "actions", while there is no concept of actions in JMPL, and there is no clear correspondence between "actions" and "states". Consequently, the translation from a Livingstone model into the syntax of MCMAS has to be performed manually, and in some circumstances the translation may be cumbersome. On the contrary, the translation from JMPL to SMV can be performed automatically using the tool `jmpl2smv`.

Nevertheless, using MCMAS allows for the expression of more complex specification patterns, in what seems to be a more natural way. In parallel with *diagnosability*, *recoverability* can also be expressed in MCMAS, whereas there does not seem to be a simple way of expressing diagnosability of recoverability with temporal-only formulae.

In general, MCMAS and other model checkers for multi-agent systems may provide more usable tools for system designers. In this paper we have investigated preliminary applications of model checking multi-agent systems for the verification of diagnosability and recoverability, and we believe that various areas may benefit from a multi-agent approach to verification, when model checkers for multi-agent systems will reach the maturity of their temporal-only counterparts.

References

[1] R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proceedings of the 10th International Confer-*

ence on Computer Aided Verification (CAV'98), volume 1427 of *LNCS*, pages 521–525. Springer-Verlag, 1998.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[3] M. Benerecetti, F. Giunchiglia, and L. Serafini. Model checking multiagent systems. *Journal of Logic and Computation*, 8(3):401–423, 1998.

[4] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking agentspeak. In J. S. Rosenschein, T. Sandholm, W. Michael, and M. Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS-03)*, pages 409–416. ACM Press, 2003.

[5] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[6] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[7] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV2: An open-source tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 359–364. Springer-Verlag, 2002.

[8] A. Cimatti, C. Pecheur, and R. Cavada. Formal verification of diagnosability via symbolic model checking. In *Proceedings of IJCAI03*, 2003.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[10] L. Console, C. Picardi, and M. Ribaudo. Diagnosis and diagnosability using pepa. In *Proceedings of the European Conference on Artificial Intelligence*, pages 131–136, Berlino, Germany, August 2000. IOS Press.

[11] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.

[12] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.

[13] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In *SPIN 2002 – Proceedings of the Ninth International SPIN Workshop on Model Checking of Software*, Grenoble, France, April 2002.

[14] W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In M. Gini, T. Ishida, C. Castelfranchi, and W. L. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 1167–1174. ACM Press, 2002.

[15] G. J. Holzmann. The model checker SPIN. *IEEE transaction on software engineering*, 23(5), 1997.

[16] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, August 2001.

[17] S. Jiang and R. Kumar. Failure diagnosis of discrete event systems with linear-time temporal logic fault specications, 2002. IEEE Trans. on Automatic Control.

[18] A. Lomuscio, T. Łasica, and W. Penczek. Bounded model checking for interpreted systems: preliminary experimental results. In M. Hinchey, editor, *Proceedings of FAABS II*, volume 2699 of *LNCS*. Springer Verlag, 2003.

[19] A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.

[20] K. McMillan. The SMV system. Technical Report CMU-CS-92-131, Carnegie-Mellon University, February 1992.

[21] W. Nabiałek, A. Niewiadomski, W. Penczek, A. Półrola, and M. Szreter. VerICS 2004: A model checker for real time and multi-agent systems. In *Proceedings of the International Workshop on Concurrency, Specification and Programming (CS&P'04)*, volume 170 of *Informatik-Berichte*, pages 88–99. Humboldt University, 2004.

[22] C. Pecheur and R. Simmons. From Livingstone to SMV: Formal verification of autonomous spacecrafts. In *Proceedings of the First Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS)*, volume 1871 of *Lecture Notes in Computer Science*. Springer Verlag, April 2000.

[23] F. Raimondi and A. Lomuscio. MCMAS - A tool for verification of multi-agent systems. http://www.cs.ucl.ac.uk/staff/f.raimondi/MCMAS/.

[24] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDD's. *Journal of Applied Logic*, 2005. To appear in Special issue on Logic-based agent verification.

[25] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, September 1995.

[26] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Transactions on Control Systems*, 4(2):105–124, March 1996.

[27] Stavros Tripakis. Fault diagnosis for timed automata. In *Proceedings of FTRTFT*, 2002.

[28] M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.