

Combine Model Checking and Runtime Verification in Multi-Agent Systems (short paper)*

Angelo Ferrando¹ and Vadim Malvone²

¹ Università degli studi di Genova, Italy
angelo.ferrando@unige.it

² Télécom Paris, France
vadim.malvone@telecom-paris.fr

Abstract. In this paper, we briefly review the history of model checking and runtime verification. We present the results obtained in the two research areas and also in their combination. Given the growing importance of the model checking on multi-agent systems, we open a door towards the combination of model checking with runtime verification over multi-agent systems.

Keywords: Model checking · Multi-agent systems · Runtime Verification.

1 Model Checking

The systems correctness is fundamental in hardware and software design, especially in the context of critical systems. With the latter, we mean systems in which failure is not an option. The main methods for software verification are: testing, simulation, and formal verification. Testing and simulation have one main issue: they can detect errors but can not determine their absence. To overcome the above problem, *formal verification* results to be very useful. This approach provides a formal-based methodology to model systems, specify properties, and verify that a system satisfies a given specification. In formal verification, the specification is usually based on temporal logics. The latter can describe the order of events without introducing the time explicitly. In temporal logics, we mainly distinguish between linear- and branching-time logics, which reflect the underlying nature of the time we consider. The most popular temporal logics are *LTL* (linear-time temporal logic) [42], *CTL* (computation tree logic) [19], and their extension *CTL** [26]. An outstanding development in the area of temporal logics has been the discovery of algorithmic methods to verify properties of finite-state systems represented by Kripke structures [35]. Hence, the formal verification of a system modelled by a Kripke structure M with respect a temporal logic specification φ can be rephrased as “Is M a model of

* Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

$\varphi?$ ”, which explains the name *model checking* (MC), as it was coined by Clarke and Emerson in [19]. Two main modalities are considered to perform MC in practice. The first option is a classical use of ad-hoc algorithms. For example, the PSPACE-COMPLETE recursive algorithms have been carried out to solve the MC problems for *LTL*. Similarly, for *CTL*, it has been shown a linear algorithm. The second option involves instead a systematic use of the automata-theoretic approach on infinite objects. In particular, a translation from a temporal logic formula φ to an automaton is provided. In this way, the MC question reduces to the emptiness problem of the intersection between the automaton corresponding to the system and the one for the complement of the property.

2 Model Checking on Multi-Agent Systems

In the Multi-Agent Systems (MAS) design and verification, temporal logics have recently assumed a prominent role for the strategic reasoning [3, 33, 17, 40, 39, 25]. Specifically, classical temporal logics have been extended to provide properties over MAS. One of the most important developments in this field is *Alternating-Time Temporal Logic* (ATL), introduced by Alur, Henzinger, and Kupferman [3]. Such a logic allows to reason about strategies of agents having the satisfaction of temporal goals as payoff criterion. More formally, it is obtained as a generalization of CTL, in which the existential E and the universal A *path quantifiers* are replaced with *strategic modalities* of the form $\langle\langle\Gamma\rangle\rangle$ and $\llbracket\Gamma\rrbracket$, where Γ is a set of *agents*. Despite its expressiveness, ATL suffers from the strong limitation that strategies are treated only implicitly in the semantics of such modalities. This restriction makes the logic less suited to formalize several important solution concepts, such as the *Nash Equilibrium*. These considerations led to the introduction of *Strategy Logic* (SL) [16, 40], a more powerful formalism for strategic reasoning. As a key aspect, this logic treats strategies as *first-order objects* that can be determined by means of the existential $\exists x$ and universal $\forall x$ quantifiers, which can be respectively read as “*there exists a strategy x* ” and “*for all strategies x* ”. Remarkably, in *SL* [40], a strategy is a generic conditional plan that at each step of the game prescribes an action. With more detail, there are two main classes of strategies: memoryless and memoryful. In the former case, agents choose an action by considering only the current game state while, in the latter case, agents choose an action by considering the full history of the game. Therefore, this plan is not intrinsically glued to a specific agent, but an explicit binding operator (a, x) allows to link an agent a to the strategy associated with a variable x . Unfortunately, the high expressivity of SL comes at a price. Indeed, it has been proved that the model-checking problem for SL becomes non-elementary complete and the satisfiability undecidable. To gain back elementariness, several fragments of SL have been considered. Among the others, *Strategy Logic with Simple-Goals* [12] considers SL formulas in which strategic operators, bindings operators, and temporal operators are coupled. It has been shown that *Strategy Logic with Simple-Goals* strictly subsume ATL and its MC problem is P-COMPLETE, as it is for ATL. To conclude this section, we want to

focus on a key aspect in MAS: the agents' visibility. Specifically, we distinguish between *perfect* and *imperfect* information games [44]. The former corresponds to a basic setting in which every agent has full knowledge about the game. However, in real-life scenarios it is common to have situations in which agents have to play without having all relevant information at hand. In computer science these situations occur for example when some variables of a system are internal/private and not visible to an external environment [36, 14]. In game models, the imperfect information is usually modelled by setting an indistinguishability relation over the states of the game [36, 44, 43]. This feature deeply impacts on the MC complexity. For example, ATL becomes undecidable in the context of imperfect information and memoryful strategies [24]. To overcome this problem, some works have either focused on an approximation to perfect information [11, 13] or developed notions of bounded memory [10].

3 Runtime verification

Runtime verification (RV) is being pursued as a lightweight verification technique bridging static verification techniques, such as MC, and testing. One of the main distinguishing features of RV is due to its nature of being performed at runtime, which opens up the possibility to act whenever incorrect behavior of a software system is detected. A fault is defined as the deviation between the current behavior and the expected behavior of the system [37, 22]. A fault might lead to a failure, but not necessarily. An error, on the other hand, is a mistake made by a human that results in a fault and possibly in a failure. Runtime verification [37] is the discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a run of a system under scrutiny satisfies or violates a given correctness property. In RV dynamic checking of the correct behavior of a system can be performed by a monitor which is generated from a formal specification of the properties to be verified. As happens for formal static verification, RV relies on a high level specification formalism to specify the expected properties of a system. Similarly to testing, RV is an effective but non exhaustive technique to verify complex properties of a system at runtime. In contrast to formal static verification and testing, RV offers opportunities for error recovery which make this approach more attractive for the development of reliable software. Not only a system can be constantly monitored for its whole lifetime to detect possible misbehavior, but also appropriate handlers can be executed for error recovery. RV ensures the system may be stopped the moment issues are identified in a tractable manner. Furthermore, the verification is not invasive, the system running should not be affected³ by the presence of the monitor, this is because the monitor does not need to generate the traces that have to be checked (in this way the state explosion problem, which is typical of the static verification, does not happen). Finally, verification can continue beyond system deployment. Similarly to MC, temporal logics are used to describe properties. Since RV works on the

³ Adding/Removing the monitor should not influence the system.

system computation at run-time, LTL properties has become predominant in this field [37, 9]. However, branching logics, such as CTL, have been recently explored as well. One can find works studying and applying μ HML (a branching-time logic with least and greatest fix points) from a runtime verification perspective [1, 31, 15]. As well as its monitorable subset MHML [6]. In the context of multiple paths, RV works on the verification of hyperproperties [21] can be found [30]. Such properties do not only check the correctness of individual traces, but can relate multiple computation traces to each other. A key example of a logic used in these scenarios is HyperLTL [20], which extends LTL with trace variables and trace quantifiers in order to refer to multiple traces at a time.

4 Runtime verification on Multi-agent Systems

In Section 2, we presented works on static verification on MAS. Here, we present the state of the art in runtime verification on MAS. In [5], the authors presented a framework to verify at runtime agent interaction protocols (AIP). The formalism used in this work allows the introduction of variables, that are then used to constrain the expected behavior in a more expressive way. In [27], the same authors proposed an approach to verify at runtime AIP using multiple monitors. This is obtained by decentralizing the global specification (specified as a Trace Expression [4]), which is used to represent the global protocol, into partial specifications denoting the single agents' perspective. In [7, 45], other works on runtime verification of agent interactions are proposed, and in [38] a framework for dynamic adaptive MAS (DAMS-RV) based on an adaptive feedback loop is presented. Other approaches to MAS RV include the proposals spin-off from the SOCS project where the SCIFF computational logic framework [2] is used to provide the semantics of social integrity constraints. To model MAS interaction, expectation-based semantics specifies the links between observed and expected events, providing a means to test runtime conformance of an actual conversation with respect to a given interaction protocol [46]. Similar work has been performed using commitments [18]. To conclude, we want to emphasize that RV has never been considered in logics for the strategic reasoning [3, 40].

5 Combination of MC and RV

Combining static and runtime verification methods raises many issues due to the fact that properties are checked against a model in the first case, and against a real running system in the second. To the best of our knowledge, very few attempts to carry out such a combination exist. In a position paper dating back to 2014, Hinrichs et al. suggested to “model check what you can, runtime verify the rest” [32]. Their work presented several realistic examples where such mixed approach would give advantages, but no technical aspects were addressed. Desai et al. [23] presented a framework to combine MC and RV for robotic applications. Kejstová et al. [34] extended an existing software model checker, DIVINE [8], with a RV mode. The system under test consists of a user program

in C or C++ together with the environment. The model checker operates in two modes: in *run* mode, a single execution of the program is explored in the standard execution order; in *verify* mode, the standard MC algorithm is applied. This extension to DIVINE is a prototype with many limitations recognized by the authors themselves. Other blended approaches exist, such as a verification-centric software development process for Java making it possible to write, type check, and consistency check behavioral specifications for Java before writing any code [47]. Although it integrates a static checker for Java and a runtime assertion checker, it does not properly integrate MC and RV. Both the Java approaches and the extension to DIVINE are targeted to specific programming languages. Finally, in [28] a recent work on using RV to validate MC assumptions is proposed. In this work, the environment is abstracted and given in input to the model checker; after that, a runtime monitor is generated and used to validate the abstraction against the running system.

6 The new challenge: combine MC and RV on MAS

In the previous sections, we presented the most recent and relevant contributions in the context of static and runtime verification. For both techniques, we also focused on their application in the MAS scenario. Finally, we cited the existing works on the combination of these two techniques. Nonetheless, to the best of our knowledge, no work combining the two approaches in the MAS context has been done, even though each one has been applied independently. Moreover, RV has been applied to MAS specifically for monitoring interaction protocols, and it has never been applied, nor considered, for checking logics for the strategic reasoning. We started this line of research in [29], where we presented a tool for combining MC and RV to find the decidability of ATL model checking in the context of imperfect information and memoryful strategies. In this work, no theoretical results are given. We are working on the theory behind it, such as complexity analysis and preservation results. On the other hand, we are also researching other ways to enrich RV for MAS properties. For example, by following the idea of predictive RV [41], we may consider to capture the predictive behavior of a monitor by verifying properties via MC on strategic properties and then apply RV for temporal properties. Another interesting and innovative approach involves the introduction of monitors to synthesize strategies. The existing logics for the strategic reasoning try to answer the question: *There exists a strategy?*. But, another important question is: *Which strategy?*. This would require to actually compute a strategy, and a monitor is a natural candidate to overcome this challenge. There are two possible ways that we are considering. The first one considers the existing relation between monitors and strategies, since they can be modelled with the same formalism (e.g. state machines). While, the other one involves strategic properties and monitors to capture the actions at runtime. In the latter line, a possible first attempt is to capture memoryless strategies in which we only need of an execution that involves all the states of the game model. Last but not least, by considering the works on RV over branching-time properties, we are evaluating the natural extension to logics for the strategic reasoning. In fact, as mentioned in the previous sections, logics for the strate-

gic reasoning, such as ATL, are generalizations of branching-time logics and by consequence a natural extension can be applied.

References

1. Aceto, L., Achilleos, A., Francalanza, A., Ingólfssdóttir, A.: A framework for parameterized monitorability. In: Baier, C., Lago, U.D. (eds.) Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10803, pp. 203–220. Springer (2018). https://doi.org/10.1007/978-3-319-89366-2_11, https://doi.org/10.1007/978-3-319-89366-2_11
2. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: The Sciff abductive proof-procedure. In: AI*IA. Lecture Notes in Computer Science, vol. 3673, pp. 135–147. Springer (2005)
3. Alur, R., Henzinger, T., Kupferman, O.: Alternating-Time Temporal Logic. *Journal of the ACM* **49**(5), 672–713 (2002)
4. Ancona, D., Ferrando, A., Mascardi, V.: Comparing trace expressions and linear temporal logic for runtime verification. In: Ábrahám, E., Bonsangue, M.M., Johnsen, E.B. (eds.) Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 9660, pp. 47–64. Springer (2016). https://doi.org/10.1007/978-3-319-30734-3_6, https://doi.org/10.1007/978-3-319-30734-3_6
5. Ancona, D., Ferrando, A., Mascardi, V.: Parametric runtime verification of multi-agent systems. In: AAMAS. vol. 17, pp. 1457–1459 (2017)
6. Attard, D.P., Francalanza, A.: A monitoring tool for a branching-time logic. In: Falcone, Y., Sánchez, C. (eds.) Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10012, pp. 473–481. Springer (2016). https://doi.org/10.1007/978-3-319-46982-9_31, https://doi.org/10.1007/978-3-319-46982-9_31
7. Bakar, N.A., Selamat, A.: Runtime verification of multi-agent systems interaction quality. In: Asian Conference on Intelligent Information and Database Systems. pp. 435–444. Springer (2013)
8. Barnat, J., Brim, L., Havel, V., Havlíček, J., Kriho, J., Lenčo, M., Ročkai, P., Štill, V., Weiser, J.: DiVinE 3.0—an explicit-state model checker for multithreaded C & C++ programs. In: International Conference on Computer Aided Verification. pp. 863–868. Springer (2013)
9. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. *J. Log. Comput.* **20**(3), 651–674 (2010). <https://doi.org/10.1093/logcom/exn075>, <https://doi.org/10.1093/logcom/exn075>
10. Belardinelli, F., Lomuscio, A., Malvone, V.: Approximating perfect recall when model checking strategic abilities. In: KR2018. pp. 435–444 (2018)
11. Belardinelli, F., Lomuscio, A., Malvone, V.: An abstraction-based method for verifying strategic properties in multi-agent systems with imperfect information. In: Proceedings of AAAI (2019)
12. Belardinelli, F., Jamroga, W., Malvone, V., Murano, A.: Strategy logic with simple goals: Tractable reasoning about strategies. In: 28th International Joint Conference on Artificial Intelligence (IJCAI 2019). pp. 88–94 (2019)

13. Belardinelli, F., Malvone, V.: A three-valued approach to strategic abilities under imperfect information. In: Proceedings of the 17th International Conference on Knowledge Representation and Reasoning. pp. 89–98 (2020)
14. Bloem, R., Chatterjee, K., Jacobs, S., Könighofer, R.: Assume-guarantee synthesis for concurrent reactive programs with partial information. In: TACAS. pp. 517–532 (2015)
15. Cassar, I., Francalanza, A.: On synchronous and asynchronous monitor instrumentation for actor-based systems. In: Cámara, J., Proença, J. (eds.) Proceedings 13th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems, FOCLASA 2014, Rome, Italy, 6th September 2014. EPTCS, vol. 175, pp. 54–68 (2014). <https://doi.org/10.4204/EPTCS.175.4>, <https://doi.org/10.4204/EPTCS.175.4>
16. Chatterjee, K., Henzinger, T., Piterman, N.: Strategy Logic. In: Concurrency Theory’07. pp. 59–73. LNCS 4703, Springer (2007)
17. Chatterjee, K., Henzinger, T., Piterman, N.: Strategy Logic. Information and Computation **208**(6), 677–693 (2010)
18. Chesani, F., Mello, P., Montali, M., Torroni, P.: Commitment tracking via the reactive event calculus. In: Proc. of the 21st International Joint Conference on Artificial Intelligence. pp. 91–96. IJCAI’09 (2009)
19. Clarke, E., Emerson, E.: Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In: Logic of Programs’81. pp. 52–71. LNCS 131, Springer (1981)
20. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Abadi, M., Kremer, S. (eds.) Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5–13, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8414, pp. 265–284. Springer (2014). https://doi.org/10.1007/978-3-642-54792-8_15, https://doi.org/10.1007/978-3-642-54792-8_15
21. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6), 1157–1210 (2010). <https://doi.org/10.3233/JCS-2009-0393>, <https://doi.org/10.3233/JCS-2009-0393>
22. Delgado, N., Gates, A.Q., Roach, S.: A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans. Softw. Eng. **30**(12), 859–872 (Dec 2004). <https://doi.org/10.1109/TSE.2004.91>, <http://dx.doi.org/10.1109/TSE.2004.91>
23. Desai, A., Dreossi, T., Seshia, S.A.: Combining model checking and runtime verification for safe robotics. In: Proc. of the 17th International Conference on Runtime Verification, RV 2017. LNCS, vol. 10548, pp. 172–189. Springer (2017)
24. Dima, C., Tiplea, F.: Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable. Tech. rep., arXiv (2011)
25. van Eijck, J.: PDL as a Multi-Agent Strategy Logic. In: Theoretical Aspects of Rationality and Knowledge’13. pp. 206–215 (2013)
26. Emerson, E., Halpern, J.: “Sometimes” and “Not Never” Revisited: On Branching Versus Linear Time. Journal of the ACM **33**(1), 151–178 (1986)
27. Ferrando, A., Ancona, D., Mascardi, V.: Decentralizing MAS monitoring with decamon. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8–12, 2017. pp. 239–248. ACM (2017), <http://dl.acm.org/citation.cfm?id=3091164>
28. Ferrando, A., Dennis, L., Cardoso, R., Fisher, M., Ancona, D., Mascardi, V.: Towards a holistic approach to verification and validation of autonomous cognitive systems. ACM Transactions on Software Engineering and Methodology (Jan 2021)

29. Ferrando, A., Malvone, V.: Strategy rv: A tool to approximate atl model checking under imperfect information and perfect recall. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. p. 1764–1766. AAMAS '21, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2021)
30. Finkbeiner, B., Hahn, C., Stenger, M., Tentrup, L.: Monitoring hyperproperties. *Formal Methods Syst. Des.* **54**(3), 336–363 (2019). <https://doi.org/10.1007/s10703-019-00334-z>, <https://doi.org/10.1007/s10703-019-00334-z>
31. Francalanza, A., Aceto, L., Ingólfssdóttir, A.: On verifying hennessy-milner logic with recursion at runtime. In: Bartocci, E., Majumdar, R. (eds.) Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22–25, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9333, pp. 71–86. Springer (2015). https://doi.org/10.1007/978-3-319-23820-3_5
32. Hinrichs, T.L., Sistla, A.P., Zuck, L.D.: Model check what you can, runtime verify the rest. In: Voronkov, A., Korovina, M.V. (eds.) HOWARD-60: A Festschrift on the Occasion of Howard Barringer's 60th Birthday, EPiC Series in Computing, vol. 42, pp. 234–244. EasyChair (2014), <https://easychair.org/publications/paper/tq7>
33. Jamroga, W., van der Hoek, W.: Agents that Know How to Play. *Fundamenta Informaticae* **63**(2-3), 185–219 (2004)
34. Kejstová, K., Rockai, P., Barnat, J.: From model checking to runtime verification and back. In: Lahiri, S.K., Reger, G. (eds.) Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13–16, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10548, pp. 225–240. Springer (2017). https://doi.org/10.1007/978-3-319-67531-2_14, https://doi.org/10.1007/978-3-319-67531-2_14
35. Kripke, S.: Semantical Considerations on Modal Logic. *Acta Philosophica Fennica* **16**, 83–94 (1963)
36. Kupferman, O., Vardi, M.: Module checking revisited. In: CAV '96. LNCS, vol. 1254, pp. 36–47. Springer-Verlag (1997)
37. Leucker, M., Schallhart, C.: A brief account of runtime verification. *The Journal of Logic and Algebraic Programming* **78**(5), 293 – 303 (2009). <https://doi.org/http://dx.doi.org/10.1016/j.jlap.2008.08.004>, <http://www.sciencedirect.com/science/article/pii/S1567832608000775>, the 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07)
38. Lim, Y.J., Hong, G., Shin, D., Jee, E., Bae, D.H.: A runtime verification framework for dynamically adaptive multi-agent systems. In: 2016 International Conference on Big Data and Smart Computing (BigComp). pp. 509–512. IEEE (2016)
39. Lorini, E.: A Dynamic Logic of Agency II: Deterministic DLA, Coalition Logic, and Game Theory. *Journal of Logic, Language, and Information* **19**(3), 327–351 (2010)
40. Mogavero, F., Murano, A., Vardi, M.: Reasoning About Strategies. In: Foundations of Software Technology and Theoretical Computer Science'10. pp. 133–144. LIPIcs 8, Leibniz-Zentrum fuer Informatik (2010)
41. Pinisetty, S., Jérón, T., Tripakis, S., Falcone, Y., Marchand, H., Preoteasa, V.: Predictive runtime verification of timed properties. *J. Syst. Softw.* **132**, 353–365 (2017). <https://doi.org/10.1016/j.jss.2017.06.060>, <https://doi.org/10.1016/j.jss.2017.06.060>

42. Pnueli, A.: The Temporal Logic of Programs. In: *Foundation of Computer Science'77*. pp. 46–57. IEEE Computer Society (1977)
43. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: *FOCS*. pp. 746–757 (1990)
44. Reif, J.H.: The complexity of two-player games of incomplete information. *JCSS* **29**(2), 274–301 (1984)
45. Roungrongsom, C., Pradubsuwun, D.: Formal verification of multi-agent system based on jade: A semi-runtime approach. In: *Recent Advances in Information and Communication Technology 2015*, pp. 297–306. Springer (2015)
46. Torroni, P., Yolum, P., Singh, M.P., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P.: Modelling interactions via commitments and expectations. In: *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI Global (2009)
47. Zimmerman, D.M., Kiniry, J.R.: A verification-centric software development process for java. In: Choi, B. (ed.) *Proceedings of the Ninth International Conference on Quality Software, QSIC 2009, Jeju, Korea, August 24-25, 2009*. pp. 76–85. IEEE Computer Society (2009). <https://doi.org/10.1109/QSIC.2009.18>, <https://doi.org/10.1109/QSIC.2009.18>