On Module Checking and Strategies

Aniello Murano
Università degli Studi di Napoli Federico II
Naples, Italy
murano@na.infn.it

ABSTRACT

Two decision problems are very close in spirit: *module checking* of CTL* and *model checking* of ATL*. The latter appears to be a natural multi-agent extension of the former, and it is commonly believed that model checking of ATL* subsumes module checking of CTL* in a straightforward way. Perhaps because of that, the exact relationship between the two has never been formally established.

A more careful look at the known complexity results, however, makes one realize that the relationship is somewhat suspicious. True, the complexities of the two problems in their most general variants match, i.e., both module checking of CTL* and model checking of ATL* are 2EXPTIME-complete. On the other hand, for the state-based fragments, module checking of CTL is EXPTIME-complete, while model checking of ATL is only P-complete. Thus, the (seemingly) less expressive framework yields significantly higher computational complexity than the (seemingly) more expressive one. This suggests that the relationship may not be as simple as believed. In this paper, we show that the difference is indeed fundamental. The way in which behavior of the environment is understood in module checking cannot be equivalently characterized in ATL*. Conversely, if one wants to embed module checking in ATL* then its semantics must be extended with two essential features, namely nondeterministic strategies and long-term commitment to strategies.

1. CTL MODULE CHECKING VS ATL MODEL CHECKING

In design and verification of formal systems, *model checking* is a well-established method to automatically check for global correctness of systems. In such a framework, in order to verify whether a system is correct with respect to a desired property, we describe its structure with a mathematical model, specify the property with a temporal logic formula, and check formally that the model satisfies the specification. This method has been first conceived for *closed systems* whose behavior is completely determined by their internal states and transitions. In this setting, models are often given as *Kripke structures* (i.e., labeled state transition graphs) and a classical temporal logic specification is CTL* or its sublogics CTL, LTL. It is worth observing that in closed models we have an internal nondeterminism. Hence, an unwinding of a Kripke structure results in an infinite tree, formally called *computation tree*, that collects all possible evolutions of the system. Then, model checking of a closed system amounts to checking whether the tree is correct with respect to the specification.

Module checking. In the last decade, interest has arisen in analyzing the behavior of individual components (or sets of components) in systems with multiple entities. The interest began in the field of reactive systems, which are systems that interact continually with their environments. In *module checking* [11], the system is modeled as a *module* that interacts with its environment, and correctness means that a desired property must hold with respect to all possible interactions. The module can be seen as a Kripke structure with states partitioned into ones controlled by the system and by the environment. Notice that the environment represents an external additional source of nondeterminism, because at each state controlled by the environment the computation can continue with any subset of its possible successor states. In other words, while in model checking we have only one computation tree to check, in module checking we have an infinite number of trees to handle, one for each possible behavior of the environment.

This makes the module checking problem harder to deal with. Indeed, while CTL* model checking is PSPACE-complete, CTL* module checking is 2EXPTIME-complete. Moreover, CTL model checking is P-complete, whereas CTL module checking is EXPTIME-complete. Finally, module checking is exponentially harder even in terms of program complexity (i.e., in case we use a fixed-size formula) where we move from LogSpace-completeness for model checking CTL to P-completeness for module checking CTL.

Alternating-time logic. Taking module checking as the starting point, researchers have looked for logics to reason about, and verify strategic behavior of agents in multi-agent systems [1, 16, 8, 6, 14]. Perhaps the most important development in this field was *alternating-time temporal logic* (ATL* for short), introduced by Alur, Henzinger, and Kupferman [1]. ATL* allows reasoning about strategies of agents with temporal goals. Formally, it is obtained as a generalization of CTL* in which the path quantifiers E (*"there exists a path"*) and A (*"for all paths"*) are replaced with *strategic modalities* of the form $\langle\langle A \rangle\rangle$ (*"A can collectively enforce that..."*), where A is a set of *agents* (a.k.a. *players*). Strategic modalities are used to express cooperation and competition among agents in order to achieve certain

Appears in: *ACTIONS 2018*

goals. In particular, they can express selective quantification over those paths that are the result of the infinite game between coalition A and the rest of agents.

Module checking vs. ATL*. Model checking of ATL* comes out as a natural multi-agent extension of CTL* module checking [1] and it is commonly believed that the latter can be embedded by the former in a straightforward way [1, 5, 18]. However, the relationship between the two has never been formally shown, which is rather remarkable given how relevant the topic is for verification of open and multi-agent systems. This lack of formal correspondence results is not without a reason, and the existing complexity results suggest potential misalignment. True, the complexity of the two problems in their most general variant match, i.e., both module checking of CTL* and model checking of ATL* are 2EXPTIME-complete. On the other hand, for the state-based fragments, we get that model checking of ATL is only P-complete while module checking of CTL is EXPTIME-complete. The (seemingly) less expressive framework yields significantly higher computational complexity than the (seemingly) more expressive one! Thus, the relationship cannot be as simple as commonly believed. Of course, there could be many reasons for this pattern of complexity (perhaps a translation to from ATL to CTL* is needed for the embedding, or the optimal translation requires exponential blowup of the formula etc.). In this paper, we show that none of those is the case, and the difference is fundamental.

Our Contribution. We show that the way in which the behavior of the environment is understood in module checking cannot be equivalently characterized in ATL*. The main reason lies in the fact that in module checking strategies of the environment are nondeterministic and irrevocable (formally represented by pruning the computation tree). In ATL*, instead, agents can only use deterministic and revocable strategies. We prove that, due to these limitations, ATL* model checking does not cover the distinguishing and expressive power of CTL* module checking, and even module checking of the less expressive logic CTL. We show that the lack of distinguishing power crucially stems from revocability of strategies in ATL*. Indeed, by considering the MIATL* extension of ATL* in which strategies are irrevocable, we show that a variant of ATL* model checking with at least the same distinguishing power as CTL* module checking. On the other hand, we show that module checking cannot be embedded in MIATL* in a natural way, because the latter lacks nondeterministic strategies. Finally, we present a syntactic and semantic variant of ATL* that exactly corresponds to the module checking of CTL* specifications.

Related work. Module checking is an active area of research. Since its introduction, it has been extensively studied in several directions. In [10], the basic question has been extended to the setting where the environment has *imperfect information* about the state of the system, showing that such a constraint does not effect the overall complexity of the problem. In [4] the module checking problem has been extended to infinite-state open systems, by considering pushdown modules. The problem has been first investigated under the perfect-information case, showing that it is exponentially harder (in comparison with the finite-state case). Then, in [2], the problem has been investigated under the imperfect-information case and proved that it is in general undecidable and that the undecidability relies on hiding information about the pushdown store. Finally, in [15] the module checking problem has been investigated with respect to bounded pushdown modules (formally *hierarchical modules*), showing up a rare case in which the program complexity of the model and module checking problems coincide.

From a more practical point of view, Martinelli [13] built a semi-automated tool to perform the finite-state module checking problem, both in the perfect and imperfect setting, with respect to a specification given in the existential fragment of CTL. Successively, an approach to CTL module checking based on tableau rules has been exploited in [3]. Godefroid and Huth also exploited an extension of module checking to reason about three-valued abstractions [7].

Literature on ATL* model checking is equally rich and the complexity of the problem has been studied in a multitude of papers. Existing implementations of ATL model checkers include **MCMAS** [12], constantly developed since 2004.

Acknowledgements. This work is based on two papers appearin in AAMAS 2014 and AAMAS 2015, respectively, done in collaboration with Wojciech Jamroga.

2. REFERENCES

[1] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.

[2] B. Aminof, A. Murano, and M. Vardi. Pushdown module checking with imperfect information. In *CONCUR'07*, LNCS 4703, pages 461–476. Springer-Verlag, 2007.

[3] S. Basu, P. S. Roop, and R. Sinha. Local module checking for ctl specifications. *ENTCS*, 176(2):125–141, 2007.

[4] L. Bozzelli, A. Murano, and A. Peron. Pushdown module checking. *Formal Meth. in Syst. Design*, 36(1):65–95, 2010.

[5] T. Brihaye, A. D. C. Lopes, F. Laroussinie, and N. Markey. ATL with strategy contexts and bounded memory. In *LFCS'09*, *LNCS 5407*, pages 92–106, Springer, 2009.

[6] K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *CONCUR'07*, pages 59–73, 2007.

[7] P. Godefroid. Reasoning about abstract open systems with generalized module checking. In *EMSOFT'03*, *LNCS 2855*, pages 223–240. Springer, 2003.

[8] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 63(2–3):185–219, 2004.

[9] M. Kacprzak and W. Penczek. Unbounded model checking for Alternating-time Temporal Logic. In *AAMAS'04*, 2004.

[10] O. Kupferman and M. Vardi. Module checking revisited. In *CAV'97*, *LNCS 1254*, pages 36–47, 1997.

[11] O. Kupferman, M. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164(2):322–344, 2001.

[12] A. Lomuscio and F. Raimondi. MCMAS : A model checker for multi-agent systems. In *TACAS'06*, *LNCS 4314*, pages 450–454, 2006.

[13] F. Martinelli. Module checking through partial model checking. Technical report, CNR Roma - TR-06, 2002.

[14] F. Mogavero, A. Murano, and M. Vardi. Reasoning about strategies. In *FSTTCS'10*, pages 133–144, 2010.

[15] A. Murano, M. Napoli, and M. Parente. Program complexity in hierarchical module checking. In *LPAR'08*, *LNCS 5330*, pages 318–332, 2008.

[16] M. Pauly. A modal logic for coalitional power in games. *Journal of Logic and Computation*, 12(1):149–166, 2002.

[17] F. Raimondi. *Model Checking Multi-Agent Systems*. PhD thesis, University College London, 2006.

[18] D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL satisfiability is indeed EXPTIME-complete. *Journal of Logic and Computation*, 16(6):765–787, 2006.

[19] D. Walther, W. van der Hoek, and M. Wooldridge. Alternating-time temporal logic with explicit strategies. In *Proceedings TARK XI'07*, pages 269–278, 2007.