EVE: A Tool for Temporal Equilibrium Analysis*

Julian Gutierrez, Muhammad Najib(✉), Giuseppe Perelli,
and Michael Wooldridge

Department of Computer Science, University of Oxford, UK
{julian.gutierrez,mnajib,giuseppe.perelli,michael.wooldridge}@cs.ox.ac.uk

Abstract. We present EVE (Equilibrium Verification Environment), a formal verification tool for the automated analysis of temporal equilibrium properties of concurrent and multi-agent systems. In EVE, systems are modelled using the Simple Reactive Module Language (SRML) as a collection of independent system components (players/agents in a game) and players' goals are expressed using Linear Temporal Logic (LTL) formulae. EVE can be used to automatically check the existence of pure strategy Nash equilibria in such concurrent and multi-agent systems and to verify which temporal logic properties are satisfied in the equilibria.

1 Introduction

We are interested in the verification of concurrent and multi-agent systems in which system components are modelled as open systems using a game-theoretic approach. In this approach, multi-agent/concurrent systems correspond to multi-player games, agents/processes to (rational) players, computation runs to plays of the game, and individual component behaviours to player strategies. Since the classical notion of correctness is not appropriate in this multi-agent setting [21], one needs different concepts to analyse such systems, and game theory provides a natural set of mathematical tools and solution concepts for that [16]. Among the proposed solution concepts, Nash equilibrium (NE) [17] is considered as the most important in non-cooperative and multi-player settings. In our framework, NE is characterised[1] as follows: given a game G, with $\mathrm{N} = \{1, \ldots, n\}$ the set of players and $\vec{a}$ a strategy profile, $\vec{a}$ is a NE if for every player $i \in \mathrm{N}$ that does not get her LTL goal formula satisfied in the play resulting from $\vec{a}$, she cannot get her goal satisfied by unilaterally changing her strategy.

In this paper, we present EVE (Equilibrium Verification Environment), which can be used to solve three key decision problems in rational synthesis and verification [9, 21]: Non-Emptiness, E-Nash, and A-Nash. These problems ask, respectively, whether a multi-player game has at least one NE, whether an LTL [18] formula holds on *some* NE, and whether an LTL formula holds on *all* NE. EVE uses a technique based on parity games to check for the existence of NE in a concurrent and multi-player game, and a model of strategies that is *memoryful*

* The authors acknowledge with gratitude the financial support of the ERC Advanced Investigator Grant 291528 ("RACE") at Oxford. Muhammad Najib is supported by the Indonesian Endowment Fund for Education (LPDP).

[1] We refer to [9, 21] for the formal characterisation of NE.

and *bisimulation invariant*. The latter property is a (desirable) key feature of our modelling framework since bisimilarity is a fundamental equivalence in concurrency which allows one to perform modular and compositional reasoning for the semantic analysis of several concurrent, reactive, and distributed systems.

There are only a few of existing tools that can be used to reason about NE in multi-player games; PRALINE [2] and MCMAS [20] are the most comparable to EVE, and yet both are different from EVE in critical ways. PRALINE does not support LTL goals and uses a model of strategies that is sensitive to bisimilar transformations, meaning that in PRALINE two games on bisimilar systems may have different sets of NE; cf., [7]. On the other hand, MCMAS can check the existence of NE in memoryless strategies only and, like PRALINE, uses a model of strategies that does not allow for bisimulation-invariant transformations, which are made, *e.g.*, when using symbolic methods via OBDDs or some model-minimisation techniques. Another tool is UPPAAL [15], which has been used to study NE in wireless networks [3]. Unlike EVE, UPPAAL works in a quantitative setting, uses Statistical Model Checking, and computes approximate NE.

2 Tool Description

Modelling Language. Each system component (agent/player) in EVE is represented as a SRML *module*, which consists of an *interface* that defines the name of the module and lists a non-empty set of Boolean variables controlled by the module, and a set of *guarded commands*, which define the choices available to the module at each state. There are two kinds of guarded commands: **init**, used for initialising the variables, and **update**, used for updating variables subsequently; we refer to [13] for further details on the semantics of SRML. In addition, we associate each module with a goal, which is specified as an LTL formula.

Implementation and Usage. EVE was developed in Python and is available online from `https://github.com/eve-mas/eve-parity`. EVE takes as input a concurrent and multi-agent system described in SRML, with player goals and a property ϕ to be checked specified in LTL. For Non-Emptiness, EVE returns "YES" (along with a set of winning players W) if the set of NE in the system is not empty, and returns "NO" otherwise. For E-Nash (A-Nash), EVE returns "YES" if ϕ holds on *some* (*all*) NE of the system, and "NO" otherwise. EVE also returns a witness for each "YES" instance as a synthesised strategy profile.

3 Case Studies

We now present two examples from the literature of distributed systems to show the practical usage of EVE. Among other things, these two examples differ in the way they are modelled as a concurrent game. While the first one is played in an arena implicitly given by the specification of the players in the game (as done in [9]), the second one is played on a graph, *e.g.*, as done in [1] with the use of concurrent game structures. Both of these modelling approaches can be used within our tool. We also use these two examples to evaluate EVE's performance in practice (and compare it against MCMAS and PRALINE) in Section 4.

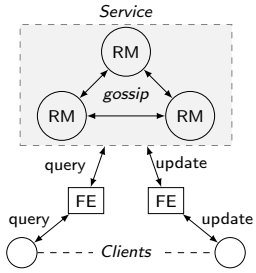

Fig. 1: Gossip framework structure.

```
module RM1 controls s1
init
:: true ~> s1':=true;
update
:: s1 ~> s1':=false;
:: s1 ~> s1':=true;
:: !s1 and (!s2 or ... or !sn)
   ~> s1':=true;
goal
:: G F (!s1);
```

Fig. 2: SRML code modelling RM_1.

Gossip Protocol. *Gossip protocols* mimic the way social networks disseminate information and have been used to solve problems in many large-scale distributed systems, such as *peer-to-peer* and *cloud* computing systems. Ladin *et al.* [14] developed a framework to provide high availability services based on the gossip approach first introduced in [4, 22]. The main feature of this framework is the use of *replica managers* (RMs) which exchange "gossip" messages periodically to keep the data updated. The architecture of such an approach is shown in Fig. 1.

We model each RM as follows: (1) When in *servicing mode*, an RM can choose either to keep in servicing mode or to switch to gossiping mode; (2) If it is in gossiping mode and there is at least another RM also in gossiping mode[2], since the information during gossip exchange is of (small) bounded size, it goes back to servicing mode in the subsequent step. The goal of each RM is to be able to gossip infinitely often. As shown in Fig. 2, the module `RM1` controls a variable: `s1`. Its value being true signifies that `RM1` is in servicing mode; otherwise in gossiping mode. Behaviour (1) is reflected in the first and second update commands, while behaviour (2) is reflected in the third update command. The goal of `RM1` is specified with the LTL formula $\mathbf{GF}\ \neg\ \mathtt{s1}$, which expresses that `RM1`'s goal is to gossip infinitely often: "always" ($\mathbf{G}$) "eventually" ($\mathbf{F}$) gossip ($\neg\,\mathtt{s1}$).

Observe that with all RMs rationally pursuing their goals, they will adopt any strategy which induces a run where each RM can gossip (with at least one other RM) infinitely often. This kind of game-like modelling gives rise to a powerful characteristic: on *all* runs sustained by a NE, the distributed system is guaranteed to have two crucial *non-starvation/liveness* properties; RMs can gossip infinitely often and clients are served infinitely often. These properties are verified in the experiments; E-Nash: no NE sustains "all RMs forever gossiping"; and with A-Nash: in all NE at least one RM is in servicing mode infinitely often.

Replica Control Protocol. Consensus is a fundamental issue in distributed computing and multi-agent systems. Gifford [6] used a quorum-based voting protocol to ensure data consistency in an information system by not allowing more than one processes in the system to read or write the same data item concurrently. To do this, each copy of a replicated data item is assigned a vote.

[2] The core of the protocol involves (at least) pairwise interactions periodically.

We can model a (modified version of) Gifford's protocol as a game as follows. The set of players $\mathrm{N} = \{1, \ldots, n\}$ in the game is arranged in a request queue represented by a sequence of states $q_1, \ldots, q_n$, where q_i means that player i is requesting to read/write the data item. At state q_i, other players in $\mathrm{N}\backslash\{i\}$ can then vote whether to allow player i to read/write. If the majority of players in N vote "yes", then the transition goes to q_0, *i.e.*, player i is allowed to read/write, and otherwise it goes to q_{i+1}[3]. The voting process then restarts from q_1. The protocol's structure is shown in Fig. 3. Notice that at the last state, q_n, there is only one outgoing arrow to q_0. The goal of each player i is to visit q_0 right after q_i infinitely often, so that the desired behaviour of the system is sustained on all NE of the system: a data item is not accessed by two processes concurrently and the data is updated in *every* round. The associated properties are verified in the experiments in Section 4. With E-Nash: there is no Nash equlibrium in which the data is never updated; with A-Nash: on all NE, each player is allowed to request to read/write infinitely often. This example uses a (deterministic) module, called "Environment", modelling the underlying concurrent game structure, shown in Fig. 3, where the game is played.

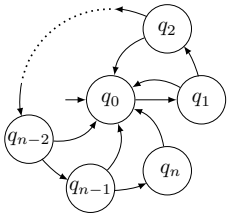

Fig. 3: Gifford's protocol modelled as a game.

4 Experimental Evaluation and Conclusions

Experiments. In order to evaluate the practical preformance of our tool and approach against MCMAS[4] and PRALINE, we present results on the temporal equilibrium analysis for the examples in Sec. 3. We ran the tools on the two examples with different numbers of players ("P"), states ("S"), and edges ("E"). The experiments were obtained on a PC with Intel i5-4690S CPU 3.20 GHz machine with 8 GB of RAM running Linux kernel version 4.12.14-300.fc26.x86_64. We report the running time[5] for solving Non-Emptiness ("ν"), E-Nash ("ϵ"), and A-Nash ("α"). For the last two problems, since there is no direct support in PRALINE and MCMAS, we used the reduction of E/A-Nash to Non-Emptiness presented in [5]. Time-out ("TO") was fixed to be 7200 seconds (2 hours).

From the experiments we observe that, in general, EVE has the best performance, followed by PRALINE and MCMAS. Although PRALINE performed better than MCMAS, both struggled (timed-out) with inputs with more than 100 edges, while EVE could handle up to about 6000 edges (for Non-Emptiness).

[3] We assume arithmetic modulo $(|\mathrm{N}| + 1)$ in this example.

[4] The tool to automatically convert EVE's input (SRML code) into MCMAS's input (ISPL code) is available online from `https://github.com/eve-mas/sevia`

[5] To carry out a fairer comparison (since PRALINE accepts Büchi objectives), we added to PRALINE's running time, the time needed to convert LTL games into its input. Translating parity games to PRALINE's input is possible in our particular examples since in those cases we can map the colours/priorities directly into Büchi condition.

Table 1: Gossip Protocol experiment results.

P	S	E	**EVE**			**PRALINE**			**MCMAS**		
			ν (s)	ϵ (s)	α (s)	ν (s)	ϵ (s)	α (s)	ν (s)	ϵ (s)	α (s)
2	4	9	0.02	0.24	0.08	0.02	1.71	1.73	**0.01**	**0.01**	**0.01**
3	8	27	0.09	0.43	0.26	0.33	26.74	27.85	**0.02**	**0.06**	**0.06**
4	16	81	**0.42**	**3.51**	**1.41**	0.76	547.97	548.82	760.65	3257.56	3272.57
5	32	243	**2.30**	**35.80**	**25.77**	10.06	TO	TO	TO	TO	TO
6	64	729	**16.63**	**633.68**	**336.42**	255.02	TO	TO	TO	TO	TO
7	128	2187	**203.05**	TO	TO	5156.48	TO	TO	TO	TO	TO
8	256	6561	**4697.49**	TO	TO	TO	TO	TO	TO	TO	TO

Table 2: Replica control experiment results.

P	S	E	**EVE**			**PRALINE**			**MCMAS**		
			ν (s)	ϵ (s)	α (s)	ν (s)	ϵ (s)	α (s)	ν (s)	ϵ (s)	α (s)
2	3	8	0.04	0.11	0.10	0.05	0.64	0.74	**0.01**	**0.01**	**0.02**
3	4	20	0.11	1.53	0.22	0.12	4.96	5.46	**0.02**	**0.06**	**0.11**
4	5	48	**0.34**	**1.73**	**0.68**	0.56	65.50	67.45	1.99	4.15	11.28
5	6	112	**1.43**	**2.66**	**2.91**	6.86	1546.90	1554.80	1728.73	6590.53	TO
6	7	256	**5.87**	**13.69**	**16.03**	94.39	TO	TO	TO	TO	TO
7	8	576	**32.84**	**76.50**	**102.12**	2159.88	TO	TO	TO	TO	TO
8	9	1280	**166.60**	**485.99**	**746.55**	TO	TO	TO	TO	TO	TO

Conclusion. We have presented EVE, a tool to analyse temporal equilibrium properties in concurrent games modelling multi-agent systems. Although there are other tools to compute pure NE (*e.g.*, PRALINE and MCMAS), they work in different settings. Moreover, while EVE uses a *richer* (bisimulation-invariant) model of strategies, it still performed better than the other two tools. In addition, this model of strategies is amenable to the use of powerful techniques for symbolic reasoning and model minimisation. Another important feature is that, in addition to Non-Emptiness, EVE has direct support for other problems in the rational verification framework [8, 9, 21], namely E-Nash and A-Nash. These two problems can be considered as counterparts to model checking in game-theoretic settings, making them very relevant in the analysis of multi-agent systems. We foresee many avenues for further work: games with imperfect information [12], quantitative payoffs [11], or branching-time goals [19, 10].

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49(5), 672–713 (Sep 2002)
2. Brenguier, R.: Praline: A tool for computing nash equilibria in concurrent games. In: CAV. pp. 890–895. Springer, Berlin, Heidelberg (2013)
3. Bulychev, P.E., David, A., Larsen, K.G., Legay, A., Mikucionis, M.: Computing nash equilibrium in wireless ad hoc networks: A simulation-based approach. In:

Proceedings Second International Workshop on Interactions, Games and Protocols, IWIGP 2012, Tallinn, Estonia, 25th March 2012. pp. 1–14 (2012)
4. Fischer, M.J., Michael, A.: Sacrificing serializability to attain high availability of data in an unreliable network. In: PODS. pp. 70–75. ACM, New York, USA (1982)
5. Gao, T., Gutierrez, J., Wooldridge, M.: Iterated boolean games for rational verification. In: AAMAS. pp. 705–713. ACM, Sao Paulo, Brazil (2017)
6. Gifford, D.K.: Weighted voting for replicated data. In: Proceedings of the Seventh ACM Symposium on Operating Systems Principles. pp. 150–162. SOSP '79, ACM, New York, NY, USA (1979)
7. Gutierrez, J., Harrenstein, P., Perelli, G., Wooldridge, M.: Nash equilibrium and bisimulation invariance. In: CONCUR. LIPIcs, vol. 85, pp. 17:1–17:16. Schloss Dagstuhl, Berlin, Germany (2017)
8. Gutierrez, J., Harrenstein, P., Wooldridge, M.: Iterated boolean games. Information and Computation 242, 53–79 (2015)
9. Gutierrez, J., Harrenstein, P., Wooldridge, M.: From model checking to equilibrium checking: Reactive modules for rational verification. Artificial Intelligence 248, 123–157 (2017)
10. Gutierrez, J., Harrenstein, P., Wooldridge, M.: Reasoning about equilibria in game-like concurrent systems. Ann. Pure Appl. Logic 168(2), 373–403 (2017)
11. Gutierrez, J., Murano, A., Perelli, G., Rubin, S., Wooldridge, M.: Nash equilibria in concurrent games with lexicographic preferences. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 1067–1073. ijcai.org (2017)
12. Gutierrez, J., Perelli, G., Wooldridge, M.: Imperfect information in reactive modules games. Information and Computation 261, 650–675 (2018)
13. van der Hoek, W., Lomuscio, A., Wooldridge, M.: On the complexity of practical atl model checking. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 201–208. AAMAS '06, ACM, New York, NY, USA (2006)
14. Ladin, R., Liskov, B., Shrira, L., Ghemawat, S.: Providing high availability using lazy replication. ACM Trans. Comput. Syst. 10(4), 360–391 (Nov 1992)
15. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell (1997)
16. Nisan, N.: Introduction to mechanism design (for computer scientists). In: Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V. (eds.) Algorithmic Game Theory, pp. 209–242 (2007)
17. Osborne, M.J., Rubinstein, A.: A Course in Game Theory (1994)
18. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57. IEEE, Rhode Island, USA (1977)
19. Toumi, A., Gutierrez, J., Wooldridge, M.: A tool for the automated verification of nash equilibria in concurrent games. In: ICTAC. LNCS, vol. 9399, pp. 583–594. Springer, Cali, Colombia (2015)
20. Čermák, P., Lomuscio, A., Mogavero, F., Murano, A.: Mcmas-slk: A model checker for the verification of strategy logic specifications. In: CAV. pp. 525–532. Springer (2014)
21. Wooldridge, M., Gutierrez, J., Harrenstein, P., Marchioni, E., Perelli, G., Toumi, A.: Rational verification: From model checking to equilibrium checking. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. pp. 4184–4191 (2016)
22. Wuu, G.T., Bernstein, A.J.: Efficient solutions to the replicated log and dictionary problems. In: Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing. pp. 233–242. PODC '84, ACM, New York, USA (1984)