EVE: A Tool for Temporal Equilibrium Analysis

Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael Wooldridge

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 represented as multi-player games. Systems are modelled using the Simple Reactive Module Language (SRML) as a collection of independent system components (players/agents in a game), which are assumed to have goals expressed using Linear Temporal Logic (LTL) formulae. EVE can be used to check the existence of Nash equilibria in such systems and 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 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 setting [18], one needs different concepts to analyse such systems, and game theory provides a natural set of mathematical tools and solution concepts for that [14]. Among the proposed solution concepts, Nash equilibrium [15] is considered as the most important in non-cooperative and multi-player settings.

In this paper, we present EVE (Equilibrium Verification Environment), a tool for temporal equilibrium analysis of concurrent and multi-agent systems represented as concurrent games. EVE solves three key decision problems in rational synthesis and verification [18, 10]: NON-EMPTINESS, E-NASH, and A-NASH, which ask, respectively, whether a multi-player game has at least one (pure-strategy) Nash equilibrium, whether an LTL formula holds on *some* Nash equilibrium, and whether an LTL formula holds on *all* Nash equilibria. EVE uses the Simple Reactive Modules Language (SRML [2]) to describe such concurrent and multi-agent systems in a succinct, high-level manner, and Linear Temporal Logic (LTL [16]) to specify individual player goals and properties to be verified of a game. EVE uses a technique based on parity games[1] to check for the existence of Nash equilibria in a concurrent and multi-player game, and a model of strategies that is *memoryful* and *bisimulation invariant*. The latter property is important because bisimilarity is one of the most fundamental features in concurrency

[1] A sketch of the main algorithm underlying EVE is provided in Appendix A.

which allows us to perform modular and compositional reasoning for the semantic analysis of several concurrent, reactive, and distributed systems.

There are only a couple of existing tools that can be used to reason about Nash equilibria in multi-player games, PRALINE [4] and MCMAS [17], both of which 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 Nash equilibria; cf., [8][2]. On the other hand, MCMAS supports model checking of Strategy Logic (SL [13]), thus making it possible to reason about Nash equilibria in games with LTL goals; however, MCMAS can check the existence of Nash equilibria in memoryless strategies only and, like PRALINE, uses a model of strategies that does not allow for bisimulation-invariant transformations, which are made, for instance, when using symbolic methods via OBDDs or some model-minimisation techniques.

2 Tool Description

Modelling Language. Systems in EVE are modelled with the *Simple Reactive Modules Language* (SRML [11]), a subset of Reactive Modules [2]. Each system component (agent/player) in SRML is represented as a *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 [11] 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 [1]. EVE takes as input a concurrent and multi-agent system described in SRML code, 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 Nash equilibria in the system is not empty, and returns "NO" otherwise. For E-Nash (A-Nash), EVE returns "YES" if ϕ holds on *some* (*all*) Nash equilibria of the system, and "NO" otherwise.

3 Case Studies

In this section, we present two examples from the literature of concurrent and 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 [10]), the second one is played on a graph, *e.g.*, as done in [3] with the use of concurrent game structures. Both of these models of games (modelling approaches) can be used within our tool. We will

[2] Experiments based on the examples in this paper are reported in Appendix B.

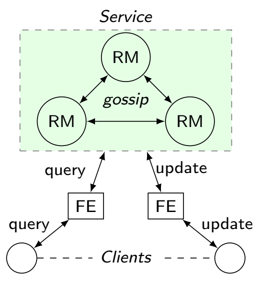

Fig. 1: Gossip framework structure.

```
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.

also use these two examples to evaluate EVE's performance in practice (and compare it against MCMAS and PRALINE) in Section 4.

Gossip Protocol. *Gossip protocols* are a class of networking and communication protocols that 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.* [12] developed a framework to provide high availability services via replication which is based on the gossip approach first introduced in [5, 19]. The main feature of this framework is the use of *replica managers* (RMs) which exchange "gossip" messages periodically in order to keep the data updated. The architecture of such an approach is shown in Fig. 1.

We can model each RM as a module in SRML 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[3], since the information during gossip exchange is of (small) bounded size, it goes back to servicing mode in the subsequent step. We then set the goal of each RM 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 **GF** $\neg$ `s1`, which expresses that `RM1`'s goal is to gossip infinitely often: "always" (**G**) "eventually" (**F**) gossip ($\neg$`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. In fact, this kind of game-like modelling gives rise to a powerful characteristic: on *all* runs that are sustained by a Nash equilibrium, the distributed system is guaranteed to have two crucial *non-starvation/liveness* properties: RMs can gossip infinitely often and clients can be served infinitely often. Indeed, these properties are verified in the experiments; E-NASH: no Nash equilibrium sustains "all RMs forever gossiping", and A-NASH: in all Nash equilibria at least one of the RM is in servicing mode infinitely often.

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

Replica Control Protocol. Consensus is a fundamental issue in distributed computing and multi-agent systems. One of the obvious domains of application is in mantaining data consistency. Gifford [7] used a quorum-based voting protocol to ensure data consistency by not allowing more than one processes to read or write a data item concurrently. To do this, each copy of a replicated data item is assigned a vote.

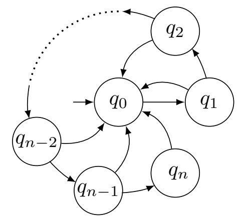

Fig. 3: Gifford's protocol modelled as a game.

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 the 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}[4]. 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. As in the previous example 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 Nash equilibria 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; E-Nash: there is no Nash equlibrium in which the data is never updated, A-Nash: in all Nash equilibria, each player is allowed to request 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.

4 Experimental Evaluation and Conclusions

Experiments. In order to evaluate the practical preformance of our tool and approach (against MCMAS 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 [6]. Time-out ("TO") was fixed to be 7200 seconds.

From the experiments we observe that in general EVE has the best performance, followed by PRALINE and MCMAS. Although PRALINE performed better

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

[5] In order to carry out a fairer comparison (since PRALINE does not accept LTL goals), we added to PRALINE's running time, the amount of time needed to convert LTL games into its input.

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

than MCMAS, both struggled (timed-out) with inputs whose edges are greater than 100, while EVE could handle up to about 6000 edges (for Non-Emptiness).

Conclusion. We have presented EVE, a tool to analyse temporal equilibrium properties in concurrent games. Although there are other tools to compute pure Nash equilibria (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 [18], 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 for the formal analysis of multi-agent systems

References

1. EVE: A tool for temporal equilibrium analysis. Available online (May 2018), `https://github.com/eve-mas/eve-parity`
2. Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods in System Design 15(11), 7–48 (Jul 1999)
3. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49(5), 672–713 (Sep 2002)

4. Brenguier, R.: Praline: A tool for computing nash equilibria in concurrent games. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. pp. 890–895. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
5. Fischer, M.J., Michael, A.: Sacrificing serializability to attain high availability of data in an unreliable network. In: Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems. pp. 70–75. PODS '82, ACM, New York, NY, USA (1982)
6. Gao, T., Gutierrez, J., Wooldridge, M.: Iterated boolean games for rational verification. In: AAMAS. pp. 705–713. ACM, Sao Paulo, Brazil (2017)
7. 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)
8. 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)
9. Gutierrez, J., Harrenstein, P., Wooldridge, M.: Expresiveness and complexity results for strategic reasoning. In: CONCUR. LIPIcs, vol. 42, pp. 268–282. Schloss Dagstuhl, Madrid, Spain (2015)
10. Gutierrez, J., Harrenstein, P., Wooldridge, M.: From model checking to equilibrium checking: Reactive modules for rational verification. Artificial Intelligence 248(Supplement C), 123 – 157 (2017)
11. 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)
12. Ladin, R., Liskov, B., Shrira, L., Ghemawat, S.: Providing high availability using lazy replication. ACM Trans. Comput. Syst. 10(4), 360–391 (Nov 1992)
13. Mogavero, F., Murano, A., Perelli, G., Vardi, M.Y.: Reasoning about strategies: On the model-checking problem. ACM Trans. Comput. Logic 15(4), 34:1–34:47 (Nov 2014)
14. 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)
15. Osborne, M.J., Rubinstein, A.: A Course in Game Theory (1994)
16. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57. IEEE, Rhode Island, USA (1977)
17. Čermák, P., Lomuscio, A., Mogavero, F., Murano, A.: Mcmas-slk: A model checker for the verification of strategy logic specifications. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification: 26th International Conference, CAV 2014. pp. 525–532. Springer International Publishing, Cham (2014)
18. 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)
19. 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, NY, USA (1984)

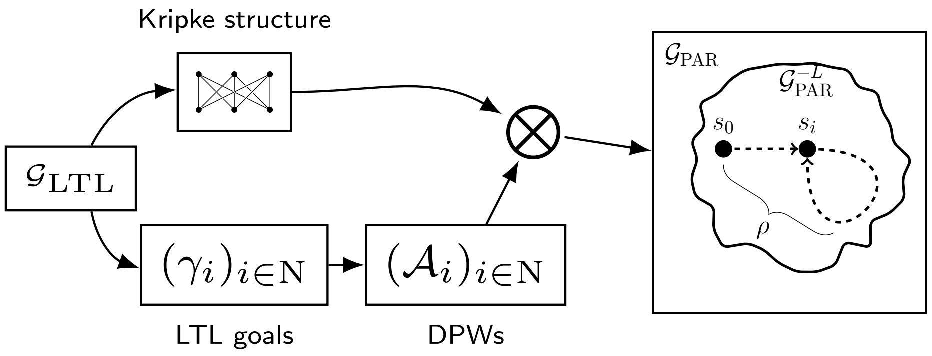
Kripke structure
$\mathcal{G}_{\text{LTL}}$
$(\gamma_i)_{i\in\text{N}}$
$(\mathcal{A}_i)_{i\in\text{N}}$
LTL goals
DPWs
$\mathcal{G}_{\text{PAR}}$
$\mathcal{G}_{\text{PAR}}^{-L}$
s_0
s_i
ρ

Fig. 4: High-level workflow of EVE.

A Main Algorithm Sketch

Temporal Equilibrium Analysis. Once a multi-agent system is modelled in SRML, it can be seen as a multi-player game in which players (the modules) use strategies to resolve the non-deterministic choices in the system. EVE uses a novel algorithm to solve Non-Emptiness via a reduction to parity games. The main idea behind this algorithm, which we will describe next, is illustrated in Fig. 4. Let $\mathcal{G}_{\text{LTL}}$ be a game, modelled using SRML, with a set of players/modules $\text{N} = \{1, \ldots, n\}$ and LTL goals $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$, one for each player. Using $\mathcal{G}_{\text{LTL}}$ we construct an associated concurrent game with parity goals $\mathcal{G}_{\text{PAR}}$ in order to shift reasoning on the set of Nash equilibria of $\mathcal{G}_{\text{LTL}}$ into the set of Nash equilibria of $\mathcal{G}_{\text{PAR}}$.

With $\mathcal{G}_{\text{PAR}}$ in our hands, we can then reason about Nash equilibria by solving a collection of parity games. As shown in [9], the existence of Nash equilibria in LTL games can be characterized in terms of punishment strategies, an idea underlying the algorithm that EVE uses. Intuitively, *punishment strategies* are strategies that prevent a player i to achieve its goal γ_i, thus eliminating any incentive of i to deviate. EVE then guesses a set of "winners" $W \subseteq \text{N}$ and computes a punishment region $\text{Pun}_j(\mathcal{G}_{\text{PAR}})$ for each $j \in L = \text{N} \setminus W$, with which a reduced parity game $\mathcal{G}_{\text{PAR}}^{-L} = \bigcap_{j\in L} \text{Pun}_j(\mathcal{G}_{\text{PAR}})$ is built. Lastly, EVE checks whether there exists a path ρ in $\mathcal{G}_{\text{PAR}}^{-L}$ that satisfies the goals of each $i \in W$. To do this, we translate $\mathcal{G}_{\text{PAR}}^{-L}$ into a deterministic Streett automata, whose language is empty if and only if so is the set of Nash equilibria of $\mathcal{G}_{\text{PAR}}$. For E-Nash problem, we simply need to find a run in the witness returned when we check for Non-Emptiness; this can be done via automata intersection[6].

B Experiments: Bisimulation Examples

These experiments are taken from the motivating examples in [8]. We extended the number of states by adding more layers to the game structures used there in

[6] For A-Nash is straightforward, since it is the dual of E-Nash.

Table 3: Example with no Nash equilibrium.

states	edges	MCMAS		EVE			PRALINE	
		time (s)	NE	time (s)	disc. time (s)	NE	disc. time (s)	NE
5	80	**0.04**	No	0.75	0.14	Yes	0.25	No
8	128	**0.24**	No	2.99	0.22	Yes	0.54	No
11	176	6.28	No	**4.80**	**0.31**	Yes	0.80	No
14	224	273.14	No	**7.46**	**0.44**	Yes	1.03	No
17	272	TO	–	**13.31**	**0.50**	Yes	1.30	No
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
50	800	TO	–	**655.80**	**2.58**	Yes	4.50	No

Table 4: Example with Nash equilibria

states	edges	MCMAS		EVE			PRALINE	
		time (s)	NE	time (s)	disc. time (s)	NE	disc. time (s)	NE
6	96	**0.02**	Yes	1.09	0.12	Yes	0.25	Yes
9	144	**0.77**	Yes	3.36	0.24	Yes	0.71	Yes
12	192	65.31	No	**7.45**	**0.40**	Yes	1.11	Yes
15	240	TO	–	**22.24**	**0.65**	Yes	1.59	Yes
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
51	816	TO	–	**1314.47**	**7.22**	Yes	8.89	Yes

order to test the practical performance of EVE, MCMAS, and PRALINE. The experiments were performed on a PC with Intel i7-4702MQ CPU 2.20GHz machine with 12GB of RAM running Linux kernel version 4.14.16-300.fc26.x86_64. We divided the test cases based on the number of Kripke states and edges; then, for each case, we report (i) the total running time ("time"), (ii) whether the tools find any Nash equilibria ("NE"), and (iii) *discounted execution time* ("disc. time"). Discounted execution time is the amount of time used after $\mathcal{G}_{\mathrm{PAR}}$ has been built until the tool terminates and outputs the result. This is to enable a comparison between EVE and PRALINE, since the latter only accepts Büchi goals (while EVE accepts LTL goals).

Table 3 shows the results of the experiments on the example in which the model of strategies that depends only on the run (sequence of states) of the game (called run-based strategies in [8]) cannot sustain any Nash equilibria, a model of strategies that is not invariant under bisimilarity. Indeed, since MCMAS and PRALINE use this model of strategies, both did not find any Nash equilibria in the game, as shown in Table 3. EVE, which uses model of strategies that, not only depends on the run of the game, but also on the actions of players (a bisimulation-invariant model of strategies called computation-based in [8]) found a Nash equilibrium in the game. We can also see that EVE outperformed MCMAS on games with 14 or more states. In fact, MCMAS timed-out[7] on games with 17 states or more, while EVE kept working efficiently for games of bigger size. We can also observe in the "disc. time" columns that PRALINE performed almost as efficiently as EVE in these experiment, although EVE performed better in both small and large instances.

In Table 4, we used the example in which Nash equilibria can be sustained using run-based strategies. As shown in the table, MCMAS found Nash equilibria in games with 6 and 9 states. However, since MCMAS uses imperfect recall, when the third layer was added (case with 12 states in Table 4) to the game, it could not find any Nash equilibria. Regarding running times, EVE outperformed MCMAS from the game with 12 states and beyond, where MCMAS timed-out on games with 15 or more states. As for PRALINE, it performed comparably to EVE in this experiment, but again, EVE performed better in all instances.

[7] We fixed the time-out value to be 3600 seconds.