Explainable Rational Synthesis in Multi-Agent Systems

Alireza Farhadi[1]
Kish International Campus, Sharif University of Technology, Iran

Mohammad Izadi
Computer Engineering Department, Sharif University of Technology, Iran

Jafar Habibi
Computer Engineering Department, Sharif University of Technology, Iran

Tobias Meggendorfer
Lancaster University Leipzig, Germany

Abstract

The synthesis of interaction protocols for multi-agent systems faces dual challenges: the double-exponential computational complexity of linear temporal logic rational synthesis, and the black-box nature of the resulting strategies, which hinders comprehension and trust. This paper introduces a novel, performant algorithm that addresses both issues. Our method improves performance by first solving a series of parity games via a *suspect game* construction to guarantee punishment for deviations, and then applying SAT-based bounded model checking to extract a minimal equilibrium path. Our implemented tool, CGES, demonstrates significant performance gains over state-of-the-art methods. Concurrently, to tackle the explainability challenge, we use a *port automata*-based connector framework for transparently modeling strategies and propose a novel algorithm for interactive, contrastive *why-not* questioning. Our work thus contributes a practical and understandable solution to rational synthesis.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Rational Synthesis, Multi-Agent Systems, Explainable AI Planning, Linear Temporal Logic, Nash Equilibrium, Port Automata, Contrastive Explanation, Constraint Solving

Digital Object Identifier 10.4230/LIPIcs.ExCoS.2025.

1 Introduction

Coordinating intelligent agents in high-stakes scenarios, such as autonomous driving, requires protocols that are both efficient and explainable. We address this by formalizing agent interactions using game theory, where agents have objectives specified in linear temporal logic (LTL). A key solution concept is the Nash equilibrium (NE), a strategy profile where no agent can benefit by unilaterally changing its strategy. However, automatically synthesizing such NEs is hindered by double-exponential complexity [1] and the black-box nature of the results. This work presents an integrated approach to overcome both hurdles. We introduce a performant synthesis algorithm and an explainability framework using port automata [2] (PA) and contrastive questioning to make rational synthesis practical and transparent.

2 Rational Synthesis Algorithm

Our approach to rational synthesis, detailed formally in Appendix A, involves transforming the input concurrent game (defined via PA) into a history-aware suspect game [3]. For each

[1] Corresponding author

relevant state Eve (the synthesizer) controls, an LTL formula φ representing punishment for deviating losers is used to form an adversarial LTL game. This is then solved as a parity game by constructing its product with a deterministic parity automaton for φ. This identifies Eve's winning states. Subsequently, a transition system based on these winning states is combined with a non-deterministic Büchi automaton for the overall desired property ψ to form a run graph. SAT-based bounded model checking is then used to find a lasso-shaped path in this run graph, representing the main part of the NE strategies. Punishment strategies are derived from Eve's winning strategies in the parity games. The CGES [2] tool implements this core synthesis algorithm. Formally proven to find valid NE paths, the algorithm, despite its double-exponential theoretical complexity (inherent to the problem), performs well in practice.

3 Enhancing Explainability

To move beyond opaque synthesis, we use PA to represent the game structure and resulting NE. PA enhances explainability through intuitive visualization as state-transition diagrams (as seen in Appendix B) and enables integration with other tools through its support for translation to various programming languages and formalisms. Our framework adopts *contrastive explanation* [4], where users can ask "Why move X instead of Y?". We propose a formal algorithm to answer such queries by analyzing the equilibrium structure, determining if the suggested change preserves the NE. While the algorithm is a key contribution, its implementation in the CGES tool is future work.

4 Evaluation and Conclusion

CGES significantly outperforms existing tools like EVE [5] in benchmarks like the gossip protocol (see Appendix C). This work offers an efficient, explainable rational synthesis method. By using suspect games, PA-based visual connectors, and a proposed algorithm for contrastive explanations, we make multi-agent system protocols more transparent and user-amendable. This contributes to explainable constraint solving, with future work focused on integrating these formal methods with large language model-based agents to guide their strategic decision-making.

References

1 Orna Kupferman and Noam Shenwald. The complexity of LTL rational synthesis. volume 25, pages 10:1–10:31, 2024.

2 Sung-Shik T. Q. Jongmans and Farhad Arbab. Overview of thirty semantic formalisms for Reo. *Sci. Ann. Comput. Sci.*, 22(1):201–251, 2012.

3 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure Nash equilibria in concurrent deterministic games. *Log. Methods Comput. Sci.*, 11(2):1–72, 2015.

4 Benjamin Krarup, Senka Krivic, Daniele Magazzeni, Derek Long, Michael Cashmore, and David E. Smith. Contrastive explanations of plans through model restrictions. *J. Artif. Intell. Res.*, 72:533–612, 2021.

5 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. *Artif. Intell.*, 287:103353, 2020.

[2] https://github.com/incaseoftrouble/cges

A Algorithm

Algorithm 1 reduces a concurrent graph game to a zero-sum, turn-based suspect game. The winning condition for Eve (the synthesizer) in this game is ensuring that any agent deviating from a proposed strategy is subsequently punished; this condition is formalized as the LTL property φ and solved via parity games to identify Eve's winning states.

A lasso-shaped equilibrium path is then extracted from a run graph using SAT-based bounded verification. This is achieved by encoding the search for a finite prefix leading to a cycle as a propositional formula. A satisfying assignment to this formula corresponds to a path where an accepting state (from the Büchi automaton for the system's overall objective) is visited infinitely often, thus guaranteeing the objectives are met and a valid NE is found.

Algorithm 1 Bounded synthesizing temporal Nash equilibrium

Inputs : PA, Agents, respective LTL objectives and payoff values
Output : A lasso-shaped path in RG_ψ and the winning strategies in PG_φ

$CG_{PA} \Longleftarrow Inputs$;
$HG_{SG} \Longleftarrow SG_{CG} \Longleftarrow CG_{PA}$ **foreach** $s \in EveStates(HG_{SG})$ **do**
  $\varphi = \bigwedge_{i \in Loser_{Agt}} \mathbf{G}(a_i \implies \neg o_i)$;
  $AG_\varphi \Longleftarrow$ sub-game of HG_{SG} reachable from s with objective φ;
  $PG_\varphi \Longleftarrow AG_\varphi \times DPA_\varphi$ **if** $Solved(PG_\varphi)$ **then**
    $s \in Winning_{Eve}$;
end
$TS_{Winning} \Longleftarrow$ initial sub-graph of HG_{SG} including $Winning_{Eve}$;
$\psi = \bigwedge_{a \in Winner_{Agents}} o_a \wedge \bigwedge_{a \in Loser_{Agents}} \neg o_a$;
$RG_\psi \Longleftarrow TS_{Winning} \times NBA_\psi$ $threshold =$ length of the longest loop-free path in RG_ψ;
for $k \leftarrow 1$ **to** $threshold$ **do**
  $\Omega_{RG_\psi}(k) \Longleftarrow RG_\psi$ **if** $SAT(\Omega_{RG_\psi}(k)) =$ *'sat'* **then**
    return indexed Boolean variables valuations ;
  else
    return "Nothing";
  end
end

In order to find a witness path of a given length k in the run graph, a Boolean constraint system is encoded and solved using a SAT-solver. This constraint system is represented by the following propositional formula $\Omega_{RG_\psi}(k)$, which encodes a sequence of length k in the Boolean representation of the run graph using a set of Boolean variables indexed up to k. The formula can then be mapped back to the initial lasso-shaped path. The SAT-solver returns an assignment for the variables if the propositional formula is true.

$$\Omega_{RG_\psi}(k) \doteq I(t^0) \wedge \bigwedge_{i=0}^{k-1} T(t^i, t^{i+1}) \wedge \bigvee_{l=0}^{k-1} \left((t^l = t^k) \wedge \bigvee_{j=l}^{k} \bigvee_{F_i \in F} F_i(t^j) \right) \tag{1}$$

The first conjunct constrains the sequence of states t^i to start from the initial state of the run graph, represented by the predicate I. The second section guarantees that successive states along the sequence satisfy the run graph's transition relation, represented by the

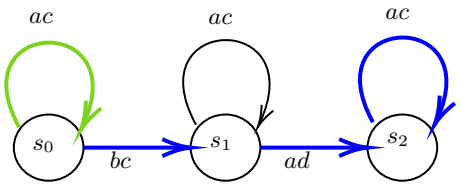

Figure 1 Concurrent game structure

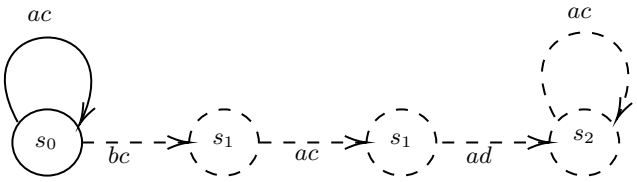

Figure 2 First Nash equilibrium for Fig.1

predicate T. The third section enforces that the sequence ends with a loop, such that at least one of the states in the loop belongs to the set of accepting states F (represented by the set of predicates F_i) of the NBA of ψ.

B An Illustrative Example

In Fig. 1, agents A_1 and A_2 interact in a concurrent game. At s_0, A_1 chooses a or b, while A_2 selects c. At s_1, A_1 must pick a, and A_2 chooses c or d. At s_2, both are restricted to a and c. A_1 aims for $\mathbf{FG}s_1$, while A_2 targets $\mathbf{FG}s_2$.

Our algorithm identifies two NEs. In the first, the system follows the green path, where neither agent achieves its objective. If A_1 deviates to s_1, A_2 redirects to s_2, blocking A_1's benefit. Fig. 2 shows the connector ensuring this equilibrium. Choosing solid transition ac at s_0 means no deviation, while dashed bc signals A_1's deviation, leading to s_1, where the protocol punishes A_1 and prevents looping.

The system follows the blue path in the second NE: A_1 fails and A_2 succeeds. Fig. 3 shows solid states and transitions for coordinated execution, while the dashed section neutralizes A_1's futile deviation. Although the blue path could loop over s_1 without affecting the payoff, the algorithm selects the shortest route.

In Fig. 2, replacing move ac with ad in the first punishment state s_1 still punishes A_1 after deviation at s_0. In Fig. 3, swapping ad with ac in any s_1 state, whether main or punishment, fails to prevent A_1 from reaching its goal.

C Implementation

The proposed algorithm is implemented in Java as a tool named Concurrent Game Equilibrium Synthesizer (CGES). The tool utilizes the Safraless LTL to DPA translation implementation based on semi-deterministic automata, providing a notable performance advantage, as handling non-deterministic models in synthesis is challenging. CGES utilizes efficient libraries

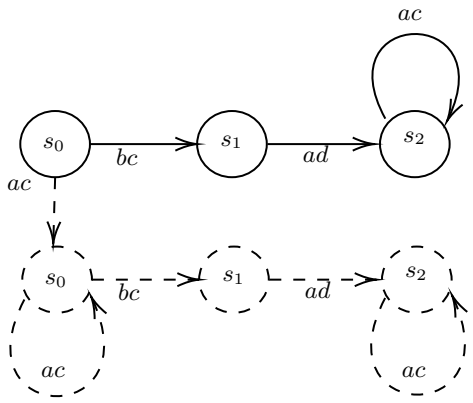

Figure 3 Second Nash equilibrium for Fig.1

Table 1 Performance evaluation of EVE and CGES in the gossip protocol

Players	2	3	4	5	6	7	8
CGES	0.2s	0.4s	0.6s	1.4s	3.8s	20.1s	167.3s
EVE	0.1s	0.2s	1.1s	13.5s	310.4s	>2 hours	?

such as Owl[3] for LTL translation, Oink[4] for parity game-solving, and Z3[5] for SAT-solving. The tool incorporates a caching mechanism to optimize performance.

To assess scalability and practical performance, we evaluate CGES on two case studies and compare it with the state-of-the-art EVE tool on an AMD Ryzen 5 3600 6-Core processor with 16 GB of RAM.

Case Study 1: Gossip Protocol

The gossip protocol is commonly used for information dissemination in large-scale systems [5]. Agents (replica managers) can be in *servicing* or *gossiping* modes, with the objective for all agents to engage in gossiping as frequently as possible. The computation of a pure Nash equilibrium becomes demanding as the number of players grows. The results of our comparison, presented in Table 1, unequivocally demonstrate CGES's significant superiority over EVE in scenarios with larger player counts.

Case Study 2: Multi-Robot Motion Planning

Multi-Robot Motion Planning (MRMP) is a fundamental challenge in robotics, requiring paths for multiple robots to reach their goals without collisions [5]. We consider a scenario where two robots must navigate an $n \times n$ grid to reach opposite corners while avoiding random obstacles. We tested on grid sizes from 3×3 to 10×10. In all tests, CGES found a valid equilibrium.

The plot in Fig. 4 shows that the average execution time for CGES grows linearly and remains under 5 seconds. In contrast, EVE's performance is reported to be exponential,

[3] https://github.com/owl-toolkit/owl
[4] https://github.com/trolando/oink
[5] https://github.com/Z3Prover/z3

exceeding 2 hours for a 10×10 grid [5]. While our approach handles increasing grid sizes well, the problem complexity still grows exponentially with the number of robots; a 4-robot example on a 10×10 grid can take 2-3 hours.

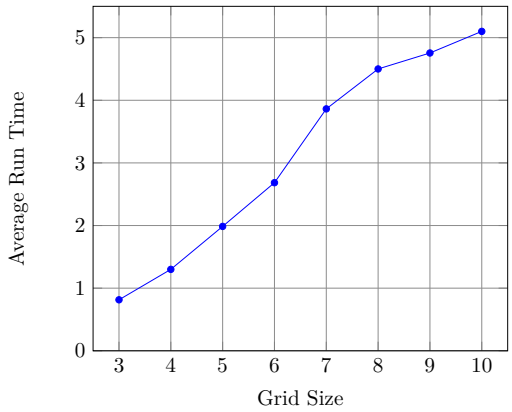

Figure 4 Plot showing the linear growth of average test execution time for CGES on MRMP case studies of increasing grid size.