Games with Additional Winning Strategies*

Vadim Malvone, Aniello Murano, and Loredana Sorrentino

Università degli Studi di Napoli Federico II

Abstract. In game theory, deciding whether a designed player wins a game corresponds to check whether he has a winning strategy. There are situations in which it is important to know whether some extra winning strategy also exists. In this paper we investigate this question over two-player finite games, under the reachability objective. We provide an automata-based technique that, given such a game, it allows to decide in linear time whether the game admits more than a winning strategy. We discuss along the paper some case studies and use them to show how to apply our solution methodology.

1 Introduction

Game theory is a very powerful mathematical framework with several useful applications in different fields. In economics, it is used to deal with solution concepts such as Nash equilibrium [18]. In computer science, it is applied to solve problems in robotics, multi-agent system verification and planning [1, 15, 20–22].

In the basic setting, a game consists of two players playing in a turn-based manner, i.e, the moves of the players are interleaved. Solving a two-player game amounts to check whether one of the players has a *winning strategy*. That is, he can use a sequence of moves (a *strategy*) that makes him to satisfy the game target, no matter how his opponent plays. In several settings, however, having instead a more precise (quantitative) answer would be beneficial. For example, in planning a rescue, it would be useful to know whether a robot team has more than a winning strategy from a critical stage, just to have a backup plan in case the scenario changes during the rescue. Such a redundancy allows to strengthen the ability of winning the game and therefore its safety.

In this paper, we address the quantitative question of checking whether a player has more than a strategy to win a two-player finite game G. We investigate this problem under the *reachability* target and show an automata-based solution to solve it in linear time. Precisely, we build an automaton that accepts only trees that are witnesses of more than one winning strategy for the designed player over the game G. Hence, we reduce the addressed quantitative question to the emptiness of this automaton. To give an evidence of our approach, we report on some cooperative and adversarial game examples.

Related works. Counting strategies has been deeply exploited in the formal verification setting [3,5,7–9,13] by means of infinite duration games. The automata construction we use here takes inspiration from the ones used in [2, 9, 19].

* This paper is partially supported by the FP7 EU project 600958-SHERPA.

2 Case Studies

In this section we consider two different case studies of two-player games. In the first case the players behave adversarial. In the second one, they are cooperative.

Cop and Robber Game. Assume we have a maze where a cop aims to catch a robber, while the latter, playing adversarial, aims for the opposite. For simplicity, we assume the maze to be a grid divided in rooms, each of them named by its coordinates in the plane (see Figure 1). Each room can have one or more doors that allow the robber and the cop to move from one room to another. Each door has associated a direction along with it can be crossed. Both the cop and the robber can enter every room. The cop, being in a room, can physically block only one of its doors. The robber can move in another room if there is a non-blocked door he can take, placed between the two rooms, with the right direction. The robber wins the game if he can reach one of the safe places (EXIT) situated in the four corners of the maze. Otherwise, the robber is blocked in a room or he can never reach a safe place, and thus the cop wins the game. We assume that both the cop and the robber are initially siting in the middle of the maze, that is in the room $(1, 1)$. Starting from the maze depicted in Figure 1, one can see that the robber has only one strategy to win the game. Consider now two orthogonal variations of the maze. For the first one, consider flipping the direction of the door d_{12}. In this case, the robber loses the game. As second variation, consider flipping the direction of the door d_6. Then the robber wins the game and he has now two strategies to accomplish it.

Escape Game. Assume we have an arena similar to the one described in the previous example, but now with a cooperative interaction between two players, a human and a controller, aiming at the same target. Precisely, consider the arena depicted in Figure 2 representing a building where a fire is occurring. The building consists of rooms and, as before, each room has one-way doors and its position is determined by its coordinates. We assume that there is only one exit in the corner $(2, 2)$. One can think of this game as a simplified version of an automatic control station that starts working after an alarm fire occurs and all doors have been closed. Accordingly, we assume that the two players play in turn and at the starting moment all doors are closed. At each control turn, he opens one door of the room in which the human is staying. The human turn consists of taking one of the doors left open if its direction is in accordance with the move. We assume that there is no communication between the players, but the move. We start the game with the human siting in the room $(0, 0)$ and the controller moving first. It is not hard to see that the human can reach the exit trough the doors d_1, d_4, d_7, d_{10} opened by the controller. Actually, this is the only possible way the human has to reach the exit. Conversely, if we consider the scenario in which the direction of the door d_3 is flipped, then there are two strategies to let the human to reach the exit. Therefore, the latter scenario can be considered as better (i.e., more robust) than the former. Clearly, this extra information can be used to improve an exit fire plan at its designing level.

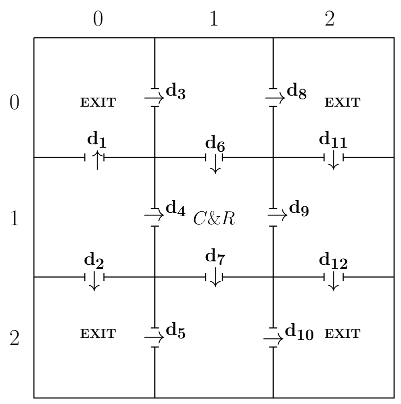

Fig. 1. Cop and Robber Game.

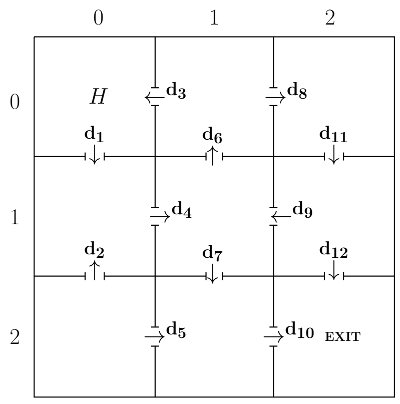

Fig. 2. Escape Game.

3 The Game Model

In this paper, we consider two-player turn-based games that are suitable to represent the case studies we have introduced in the previous section. Precisely, we consider games consisting of an arena and a target. The arena describes the configurations of the game through a set of states, being partitioned between the two players. In each state, only the player that owns it can take a move. This kind of interaction is also known as token-passing. About the target, we consider the reachability objective, that is some states are declared *target*. A winning strategy for a designed player is a path from the initial state to a target state. If such a winning strategy exists we say that the player wins the game. Clearly, the player has more than a winning strategy if there are different paths reaching a target state. The formal definition of the considered game model follows.

Definition 1. *A* turn-based two-player reachability game *(2TRG, for short), played between Player* 0 *and Player* 1*, is a tuple* $G \triangleq < \mathrm{St},\ s_I,\ \mathsf{tr},\ \mathrm{W} >$*, where* St *is a finite non-empty set of* states*, partitioned in* St_0 *and* St_1 *with* St_i *being the set of states of Player i,* $s_I \in \mathrm{St}$ *is a designated* initial state*,* W *is a set of target states, and* $\mathsf{tr} \subseteq \mathrm{St}_i \times \mathrm{St}_{1-i}$*, for* $i \in \{0,1\}$ *is a* transition function *mapping a state of a player to a state belonging to the other player.*

The previous two case studies can be easily modeled using a 2*TRG*. We now give some details. As set of states we use all the rooms in the maze, together with the status of their doors. For example, the state $((0,0),\{d_1^c, d_3^c\})$ is the initial state of the *Escape Game* where d_i^c means that the door d_i is closed. For an open door, instead, we will use the label o in place of c. Formally, let $D_{i,j}$ be the set of doors (up to four) belonging to the room (i,j), which can be flagged either with c (closed) or o (open), then we set $\mathrm{St} \subseteq \{((i,j), D_{i,j}) \mid 0 \leq i,j \leq 2\}$. Transitions are taken by the human/robber in order to change the room (coordinates) or by the cop/controller to change the status of its doors. These moves are taken in accordance with the shape of the maze. The partitioning of the states between the players follows immediately, as well as the definition of the target states.

4 Searching for Multiple Winning Strategies

To check whether Player 1 has a winning strategy in a 2*TRG* G one can use a classic backward algorithm. We briefly recall it. Let *succ* : $\mathrm{St} \rightarrow 2^{\mathrm{St}}$ be the function that for each state $s \in \mathrm{St}$ in G gives the set of its successors. The algorithm starts from a set S equal to W. Iteratively, it tries to increase S by adding all states $s \in \mathrm{St}$ that satisfy the following conditions: (i) $s \in \mathrm{St}_0$ and $succ(s) \subseteq S$; or, (ii) $s \in \mathrm{St}_1$ and $succ(s) \cap S \neq \emptyset$. If S contains at a certain point the initial state, then Player 1 wins the game.

In case one wants to ensure that more than a winning strategy exists, the above algorithm becomes not appropriate. We use instead a top-down automata-theoretic approach. To give an intuition of this solution, first consider that in a 2*TRG* a *witness* for a winning strategy is a tree that takes for each node corresponding to a state s in the game, one successor if s belongs to Player 1, or all successors, otherwise. Indeed, if all the leaves of this tree are target states, then surely Player 1 has a winning strategy over the game. In case we want to ensure that at least two winning strategies exist then at a certain point along the tree Player 1 must take two successors. We build a tree automaton that accepts exactly this kind of witness trees. For the lack of space, we omit the definition of tree and the related concepts. We refer for this to [19]. We now give a definition of witness trees and then we define the tree automata required.

Definition 2. *Given a* 2*TRG* G *a witness tree* T *is a finite tree whose nodes correspond to the states of* G *and satisfy the following properties:*

- *the root node corresponds to the initial state* s_I *of* G*;*
- *for each* $x \in T$ *that is not a leaf and corresponds to a Player* 1 *state* s *in* G*, it holds that* x *has as children a non-empty subset of* $succ(s)$*;*
- *for each* $x \in T$ *that is not a leaf and corresponds to a Player* 0 *state* s *in* G*, it holds that* x *has as children the set of* $succ(s)$*;*
- *each leaf of* T *corresponds to a target state in* G*;*
- *each leaf in* T *has an ancestor node* x *that corresponds to a Player* 1 *state in* G *and it has at least two children.*

The above definition, but the last item, is the classical characterization of strategy tree. Last property further ensures that Player 1 has the ability to enforce at least two winning strategies no matter how Player 0 acts.

Definition 3. *A nondeterministic tree automaton (NTA, for short) is a tuple* $A \triangleq < Q, \Sigma, q_0, \delta, F >$*, where* Q *is a set of states,* Σ *is an alphabet,* $q_0 \in Q$ *is an initial state,* $\delta : Q \times \Sigma \rightarrow 2^{Q^*}$ *is a transition function mapping pairs of states and symbols to a set of tuples of states, and* $F \subseteq Q$ *is a set of the accepting states.*

An *NTA* A recognizes trees and works as follows. For a node tree labeled by σ and A being in a state s, it sends different copies of itself to successors in accordance with δ. For example, if $\delta(s, \sigma) = \{(s_1, s_2), (s_3, s_4)\}$ either A proceeds with a left child in state s_1 and a right child in state s_2, or it proceeds with a left child in state s_3 and a right child in state s_4. By $L(A)$ we denote the set of trees accepted by A. It is not empty if $L(A) \neq \emptyset$.

We now give the main result of this paper, *i.e.* we show that it is possible to decide in linear time whether, in a 2*TRG*, Player 1 has more than a winning strategy. We later report on the application of this result along the case studies.

Theorem 1. *For a* 2*TRG game* G *it is possible to decide in linear time whether Player* 1 *has more than a strategy to win the game.*

Proof (sketch). Consider a 2*TRG* game G. We build an *NTA* A that accepts all trees that are witnesses of more than a winning strategy for Player 1 over G. We briefly describe the automaton. It uses $\mathrm{St} \times \{ok, split\}$ as set of states where *ok* and *split* are flags and the latter is used to remember that along the tree Player 1 has to ensure the existence of two winning strategies by opportunely choosing a point where to "split". We use a one-letter alphabet Σ, as this set takes no role. For the initial state we set $q_0 = (s_I, split)$. For the transitions, starting from a state $q = (s, flag)$, we distinguish between two cases: *(i)* $s \in \mathrm{St}_0$. If $flag = ok$ then $\delta(q) = succ(s) \times \{ok\}$, otherwise, let $succ(s) = \{s_1, \ldots, s_n\}$ then $\delta(q) = \{((s_1, f_1), \ldots, (s_n, f_n))\}$ and there exists $1 \leq i \leq n$ such that $f_i = split$ and for all $j \neq i$, we have $f_j = ok$. *(ii)* $s \in \mathrm{St}_1$. If $flag = ok$ then $\delta(q) = \{((s', ok))\}$ with $s' \in succ(s)$, otherwise, we have $\delta(q) = \{((s', ok), (s'', ok)), ((s', split))\}$, with $s', s'' \in succ(s)$ and $s' \neq s''$. The set of accepting states is $\mathrm{W} \times \{ok\}$. A tree is accepted by A if at a certain point Player 1 can take two successors in G both leading to a target state.

The size of the automaton is just linear in the size of the game. Moreover, by using the fact that, from [19], checking the emptiness of an *NTA* can be performed in linear time, the desired complexity result follows. □

Consider the Escape Game example. By applying the above construction, the automaton A accepts an empty language. Indeed, for each input tree, A always leads to a leaf containing either a state with a non-target component (i.e., the tree is a witness of a losing strategy) or with a flag *split* (i.e., Player 1 cannot select two winning strategies). Conversely, consider the same game, but flipping the direction of the door d_3 in the maze. In this case, A accepts exactly one tree. Indeed starting from the initial state $(((0,0), \{d_1^c, d_3^c\}), split)$, A proceeds in two different direction with states $(((0,0), \{d_1^o, d_3^c\}), ok)$ and $(((0,0), \{d_1^c, d_3^o\}), ok)$, that refer to two distinct winning strategies for the controller.

A similar reasoning can be exploited with the Cop and Robber Game example. Indeed, the automaton accepts an empty language. Conversely, by flipping the door d_4, it accepts the tree that is witnessing of two different winning strategies each of them going through one of the two doors left unblocked by the cop.

We finally conclude this section by recalling that in one player games the problem of checking whether more than a winning strategy exists can be checked in NLogSpace. Indeed, it is enough to extend the classic *path algorithm* by extending it in a way that we search for two paths by doubling the used logarithmic working space [1].

[1] We thank prof. Alberto Pettorossi for useful discussion in this respect.

5 Conclusion and Future Work

In this paper we have introduced a simple but effective automata-based methodology to check whether a player has more than a winning strategy in a two-player game under the reachability objective. We have showed how this methodology can be applied in practice by reporting on its use over two different game scenarios, one cooperative and one adversarial. We believe that the solution algorithm we have conceived in this paper can be used as core engine to count strategies in more involved game scenarios and in many solution concepts reasoning.

This work opens to several interesting questions and extensions. For instance, it would be worth investigating game scenarios in which one or both players have imperfect information regarding some moves of the other player. The imperfect information setting is an important field of study in game theory with several practical applications. For some related works see [6, 12, 14]. Another interesting direction would be to consider the counting of strategies in multi-agent concurrent games. This kind of games have several interesting applications in artificial intelligence [20–22]. One can also consider some kind of hybrid game, where one can opportunely combine team of players working concurrently with some others playing in a turn-based manner [10, 11, 17]. Last but not least, it would be worth investigating infinite-state games. These games arise for example in case the interaction among the players behaves in a recursive way [4, 16].

References

1. R. Alur, T. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002.
2. A. Bianco, F. Mogavero, and A. Murano. Graded Computation Tree Logic. In *LICS'09*, pages 342–351. IEEE Computer Society, 2009.
3. P. Bonatti, C. Lutz, A. Murano, and M. Vardi. The Complexity of Enriched muCalculi. *LMCS*, 4(3):1–27, 2008.
4. L. Bozzelli, A. Murano, and A. Peron. Pushdown Module Checking. *FMSD*, 36(1):65–95, 2010.
5. D. Calvanese, G. D. Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *IJCAI'99*, volume 99, pages 84–89, 1999.
6. K. Chatterjee, L. Doyen, T. A. Henzinger, and J. Raskin. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3(4):1–23, 2007.
7. M. Faella, M. Napoli, and M. Parente. Graded Alternating-Time Temporal Logic. *FI*, 105(1-2):189–210, 2010.
8. A. Ferrante and A. Murano. Enriched Mu-Calculi Module Checking. In *FOSSACS'09*, LNCS 5504, pages 183–197. Springer, 2007.
9. A. Ferrante, A. Murano, and M. Parente. Enriched Mu-Calculi Module Checking. *LMCS*, 4(3):1–21, 2008.
10. W. Jamroga and A. Murano. On Module Checking and Strategies. In *AAMAS'14*, pages 701–708. IFAAMAS, 2014.
11. W. Jamroga and A. Murano. Module checking of strategic ability. In *AAMAS'15*, pages 227–235. IFAAMAS, 2015.

12. J.H. Reif. The complexity of two-player games of incomplete information. *Journal of computer and system sciences*, 29(2):274–301, 1984.
13. O. Kupferman, U. Sattler, and M. Vardi. The Complexity of the Graded muCalculus. In *CADE'02*, LNCS 2392, pages 423–437. Springer, 2002.
14. O. Kupferman and M. Vardi. Module Checking Revisited. In *CAV'97*, LNCS 1254, pages 36–47. Springer, 1997.
15. O. Kupferman, M. Vardi, and P. Wolper. Module Checking. *IC*, 164(2):322–344, 2001.
16. A. Murano and G. Perelli. Pushdown multi-agent system verification. In *IJCAI'15*, 2015.
17. A. Murano and L. Sorrentino. A game-based model for human-robots interaction. In *WOA'15*, CEUR Workshop Proceedings. CEUR-WS.org. To appear, 2015.
18. R. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, 1991.
19. W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 133–192. Elsevier and MIT Press, 1990.
20. M. Wooldridge. Intelligent Agents. In G. Weiss, editor, *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence.* MIT Press: Cambridge, Mass, 1999.
21. M. Wooldridge. *Reasoning about Rational Agents.* MIT Press : Cambridge, Mass, 2000.
22. M. Wooldridge. *An Introduction to Multi Agent Systems.* John Wiley & Sons, 2002.