Fundamenta Informaticae XX (2017) 1–20
DOI 10.3233/FI-2015-0000
IOS Press

Additional Winning Strategies in Reachability Games*†

Vadim Malvone
Università degli Studi di Napoli Federico II
vadim.malvone@unina.it

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

Loredana Sorrentino
Università degli Studi di Napoli Federico II
loredana.sorrentino@unina.it

Abstract. In game theory, deciding whether a designed player wins a game amounts to check whether he has a winning strategy. However, there are several game settings in which knowing whether he has more than a winning strategy is also important. For example, this is crucial in deciding whether a game admits a *unique* Nash Equilibrium, or in planning a rescue as this would provide a backup plan.

In this paper we study the problem of checking whether, in a two-player reachability game, a designed player has more than a winning strategy. We investigate this question both under perfect and imperfect information about the moves performed by the players. We provide an automata-based solution that results, in the perfect information setting, in a linear-time procedure; in the imperfect information setting, instead, it shows an exponential-time upper bound. In both cases, the results are tight.

Keywords: Additional Strategies; Two-Player Reachability Games; Graded Modalities; Imperfect Information

Address for correspondence: Università di Napoli Federico II, Department DIETI, Via Claudio, n.21, 80125, Napoli, Italy.
*Partially supported by the GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi.
†This is an extended version of the work [1] appearing in the Proc. of CILC 2015, the 30th Italian Conf. on Computational Logic.

1. Introduction

Game theory is a powerful mathematical framework to reason about *reactive* systems [2]. Over the years, it has been usefully applied in several different domains. In economics, it is used to deal with solution concepts such as Nash equilibrium [3]. In biology, it is used to reason about the *phenotypic evolution* [4]. In computer science, it is applied to solve problems in robotics, multi-agent system verification, synthesis, and planning [5, 6, 7].

In the basic setting, a (finite) game consists of two players, conventionally named $Player_0$ and $Player_1$, playing a finite number of times, in a turn-based manner, i.e., the moves of the players are interleaved. Technically, the configurations (states) of the game are partitioned between $Player_0$ and $Player_1$ and a player moves in a state whenever he owns it. Solving a two-player game amounts to check whether $Player_0$ has a *winning strategy*. That is, to check whether he can take a sequence of move *actions* (a *strategy*) that allows him to satisfy the game objective, no matter how his opponent plays.

Depending on the visibility the players have over the action moves performed by their opponents, we distinguish between *perfect* and *imperfect* information games. In the former case, both players have full knowledge of the evolution of the game, in every moment. This may not be possible in the latter case where players often have to come to decisions without having all relevant information at hand. Both settings have been largely investigated in the literature with several real-life applications. A classic approach suitable to model both is to make use of a relation of indistinguishability on the action moves/configurations of the arena [8, 9, 10, 11, 12]. In this case, during a play, it may happen that a player cannot tell precisely in which state he is, but rather he observes a set of states. This means that over indistinguishable scenarios a player is forced to use the same strategy. Straightforwardly, the perfect information setting corresponds to just using the identity relation.

In several game settings, it is mandatory to have a more precise (quantitative) information about *how many* winning strategies a player has at his disposal. For example, in Nash Equilibrium, such an information amounts to solve the challenging question of checking whether the equilibrium is unique [13, 14, 15, 16, 17, 18, 19, 20, 21]. This problem impacts on the predictive power of Nash Equilibrium since, in case there are multiple equilibria, the outcome of the game cannot be uniquely pinned down [22, 23, 24]. As another example, consider the setting of robot rescue planning [25, 26, 27]. It is not hard to imagine situations in which it is vital to know in advance whether a robot team has more than a winning strategy from a critical stage, just to have a backup plan in case an execution of a planned winning strategy cannot be executed anymore. Such a redundancy allows to strengthen the ability of winning the game and, specifically, the rescue capability.

In this paper, we address the quantitative question of checking whether $Player_0$ has more than a strategy to win a finite two-player game G. We investigate this problem under the *reachability* objective, *i.e.* some states of the game arena are declared *target*. We consider both the cases in which the players have perfect or imperfect information about the moves performed by their opponent. We solve the addressed problem by using an automata-theoretic approach. 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. Our automata solution mainly consists in extending the classic approaches by further individuating a place where $Player_0$ has the ability to follow two different ways (i.e., strategies) to reach a target state. While this may look simple in the perfect information setting, in the imperfect case it requires some careful thoughts. Furthermore, in support to the technical contribution of our solution we observe the following: *(i)* it is an use of automata and an extension of previous approaches never explored before; *(ii)* it provides an elegant solution ; *(iii)* it is an optimal solution as it gives a tight upper bound, *(iv)* it is an easy scalable solution, as one can easily inject more sophisticated solution concepts such as safety, fairness, etc.

By means of the automata-theoretic approach, one can also check for other "forms" of additional winning conditions. For example one can check whether $Player_0$ can win against *all but one* $Player_1$ *strategies*. This is intimately related to the concept of *almost surely-winning* in probabilistic games [28]. As a practical application, this is useful in game design as it can highlight the presence of a unique undesired behavior of the adversarial player and possibly suggest a way to prevent it. Similarly, it is useful in security; for example it can highlight a flow in a firewall (a successful attack coming from the environment) and suggest a way to correct it. Technically, the solution to the question "Does $Player_0$ beat all $Player_1$ strategies but one?" reduces to first build an automaton that collects all tree strategies for $Player_0$ that, except for one path, they correspond to winning strategies, and then check for its non-emptiness.

In a broader vision, the importance of our work resides on the fact that it can be seen as a core engine and as a first step through the efficient solution of important problems in computer science and AI. Among the others, we mention checking the uniqueness of Nash Equilibrium under imperfect information for reachability targets. This field has received much attention recently and some results can be found in top venues such as [29, 30]. However, all the approaches used in the mentioned papers lead to a non-elementary complexity, as they are shaped for very reach strategic formalisms to represent the solution concepts, and thus far beyond the tight complexity we achieve instead in this work.

Along the paper we make use of some cooperative and adversarial game examples that will help to better explain the specific game setting we are studying and the solution approaches we provide.

Related works. Counting strategies has been deeply exploited in the formal verification of *reactive* systems by means of specification logics extended with *graded modalities*, interpreted over games of infinite duration [31, 32, 20, 21]. However, our work is the first to consider additional winning strategies in the imperfect information setting. Also, it is worth recalling that, on the perfect information side, the solution algorithms present in the literature for graded modalities [20, 21] have been conceived to address complicated scenarios and, consequently, they usually perform much worse (*w.r.t.* the asymptotic complexity) than our algorithm on the restricted setting we consider. Clearly one can express with graded modalities the existence of additional strategies in a game. To see how this is possible we refer to [21]for an example in the perfect information setting.

Graded modalities have been first investigated over *closed* systems, i.e., one-player games, to count moves and paths in system models. A pioneering work is [33], where these modalities have been studied in classic modal logic. Successively, they have been exported to the field of *knowledge representation*, to allow quantitative bounds on the set of individuals satisfying specific properties, as well as they have been investigated in first-order logic and description logic. Specifically, they are known as *counting quantifiers* in first-order logics [34], *number restrictions* in *description logics* [35, 36, 37, 38] and *numerical constraints* in query languages [38, 39]

In [40], *graded* μCALCULUS has been introduced in order to express and evaluate statements about a given number of immediately accessible worlds. Successively in [41], the notion of graded modalities have been extended to deal with number of paths. Among the others graded CTL (GCTL, for short) has been introduced with a suitable axiomatization of counting [41]. That work has been recently extended in [42] to address GCTL*, a graded extension of CTL*.

In this work we analyze and compare different strategies in two-player games. The comparison between strategies is a problematic that has been intensively investigated in other works. Among the others, we mention [43], where the concept of *permissive strategies* has been introduced. However, the aim of that paper is to compare strategies in order to come up with a single strategy that allows to represent all of them by one.

In multi-player system verification, we also witness several specific approaches to count strategies. Chronologically, we first mention *module checking for graded* μCALCULUS [32], where the counting is restricted to moves in a two-player setting. Then, in [20, 21], motivated by counting Nash equilibria, two different graded extension of Strategy Logic have been considered.

We finally remark that the automata-theoretic solution we provide takes inspiration from the ones used in [8, 32, 41, 44, 45]. In details, in [8] such a technique is used to show that the problem is EXPTIME-complete *w.r.t.* CTL formulas and 2EXPTIME-complete *w.r.t.* CTL* formulas. In [32], an automata-theoretic approach is used to show that the same problem over pushdown structures and graded μCALCULUS formulas is 2EXPTIME-complete. In [41], automata are used to show that graded CTL formulas are satisfiable in exponential time. In [44] efficient algorithms for the emptiness problem of word and tree automata are provided. Finally, in [45], alternating tree automata are used in the imperfect information case on the synthesis problem. However, our solution is much more efficient since it is directly constructed for the simpler setting of two-player turn-based games of finite duration, played with respect to the reachability objective.

Outline. The sequel of the paper is structured as follows. In Section 2 we introduce some preliminary concepts. In Section 3 we describe two examples that are useful to introduce our game setting. In Section 4 we introduce the definition of game under perfect information and in Section 5 we solve the additional winning strategies problem by means of an automata-theoretic approach. In Section 6 we consider imperfect information and in Section 7 we give some solutions for this game setting. We conclude with Section 8 in which we report some discussions and suggest some directions for future work.

2. Preliminaries

In this section we introduce some preliminary concepts needed to properly define the game setting under exam as well as to describe the adopted solution approach. In particular, we introduce trees useful to represent strategies and automata to collect winning strategies.

Trees. Let Υ be a set. An Υ*-tree* is a prefix closed subset $T \subseteq \Upsilon^*$. The elements of T are called *nodes* and the empty word ε is the *root* of T. For $v \in T$, the set of *children* of v (in T) is $child(T, v) = \{v \cdot x \in T \mid x \in \Upsilon\}$. Given a node $v = y \cdot x$, with $y \in \Upsilon^*$ and $x \in \Upsilon$, we define $anc(v)$ to be y, *i.e.*, the ancestors of v, and $last(v)$ to be x. We also say that v *corresponds* to x. The complete Υ-tree is the tree Υ^*. For $v \in T$, a (full) path π of T from v is a *minimal* set $\pi \subseteq T$ such that $v \in \pi$ and for each $v' \in \pi$ such that $child(T, v') \neq \emptyset$, there is exactly one node in $child(T, v')$ belonging to π. Note that every word $w \in \Upsilon^*$ can be thought of as a path in the tree Υ^*, namely the path containing all the prefixes of w. For an alphabet Σ, a Σ-labeled Υ-tree is a pair $<T, V>$ where T is an $\Upsilon-$tree and $V : T \to \Sigma$ maps each node of T to a symbol in Σ.

Automata Theory. We now recall the definition of *alternating tree automata* and its special case of *nondeterministic tree automata*[46, 47, 48].

Definition 2.1. An *alternating tree automaton* (ATA, for short) is a tuple $A = < \Sigma, D, Q, q_0, \delta, F >$, where Σ is the alphabet, D is a finite set of directions, Q is the set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(D \times Q)$ is the transition function, where $\mathcal{B}^+(D \times Q)$ is the set of all positive Boolean combinations of pairs (d, q) with d direction and q state, and $F \subseteq Q$ is the set of the accepting states.

An ATA A recognizes (finite) trees by means of (finite) runs. For a Σ-labeled tree $< T, V >$, with $T = D^*$, a run is a $(D^* \times Q)$-labeled N-tree $< T_r, r >$ such that the root is labeled with (ε, q_0) and the labels of each node and its successors satisfy the transition relation.

For example, assume that A, being in a state q, is reading a node x of the input tree labeled by ξ. Assume also that $\delta(q, \xi) = ((0, q_1) \vee (1, q_2)) \wedge (1, q_1)$. Then, there are two ways along which the construction of the run can proceed. In the first option, one copy of the automaton proceeds in direction 0 to state q_1 and one copy proceeds in direction 1 to state q_1. In the second option, two copies of A proceed in direction 1, one to state q_1 and the other to state q_2. Hence, $\vee$ and $\wedge$ in $\delta(q, \xi)$ represent, respectively, choice and concurrency. A run is *accepting* if all its leaves are labeled with accepting states. An input tree is accepted if there exists a corresponding accepting run. By $L(A)$ we denote the set of trees accepted by A. We say that A is not empty if $L(A) \neq \emptyset$.

As a special case of alternating tree automata, we consider *nondeterministic tree automata* (NTA, for short), where the concurrency feature is not allowed. That is, whenever the automaton visits a node x of the input tree, it sends to each successor (direction) of x at most one copy of itself. More formally, an NTA is an ATA in which δ is in disjunctive normal form, and in each conjunctive clause every direction appears at most once.

3. Case Studies

In this section we introduce two different case studies of two-player games. In the first case the players behave adversarial. In the second one, they are cooperative. These running examples are useful to better understand some technical parts of our work.

3.1. 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 in every room. The cop, being in a room, can physically block only one of its doors or can move in another room. 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 $(1, 1)$. It important to note that the game is played in a turn-based manner, and the cop is the first player

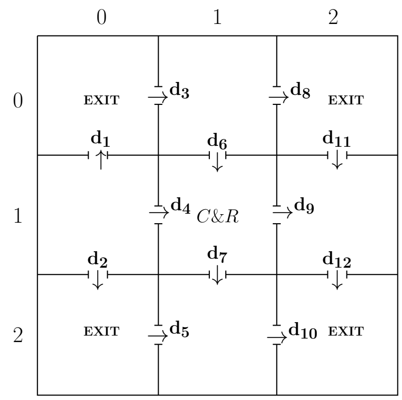

Figure 1: Cop and Robber Game.

that can moves. Starting from the maze depicted in Figure 1, one can see that the robber has only one strategy to win the game. In fact, if the cop blocks the door d_7 (*resp.*, d_9), the robber can choose the door d_9 (*resp.*, d_7), then the cop can go in the room $(1, 2)$ (*resp.*, $(2, 1)$), and finally the robber can choose the door d_{12} (*resp.*, d_{10}) and then wins 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.

3.2. 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. 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.

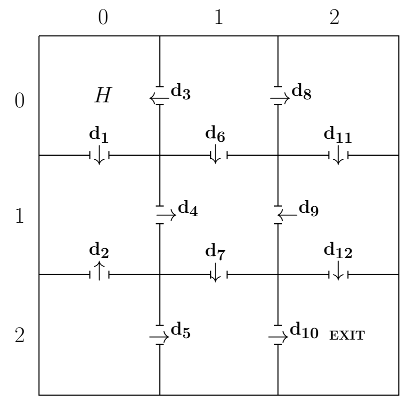

Figure 2: Escape Game.

4. The Game Model

In this section, 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*. The formal definition of the considered game model follows.

Definition 4.1. A *turn-based two-player reachability game* ($2TRG$, for short), played between $Player_0$ and $Player_1$, is a tuple $G \triangleq <\mathrm{St},\ s_I,\ \mathrm{Ac},\ \mathtt{tr},\ \mathrm{W}>$, where $\mathrm{St} \triangleq \mathrm{St}_0 \cup \mathrm{St}_1$ is a finite non-empty set of *states*, with St_i being the set of states of $Player_i$, $s_I \in \mathrm{St}$ is a designated *initial state*, $\mathrm{Ac} \triangleq \mathrm{Ac}_0 \cup \mathrm{Ac}_1$ is the set of actions, W is a set of target states, and $\mathtt{tr} : \mathrm{St}_i \times \mathrm{Ac}_i \to \mathrm{St}_{1-i}$, for $i \in \{0, 1\}$ is a *transition function* mapping a state of a player and its action to a state belonging to the other player.

To give the semantics of $2TRG$s, we now introduce some basic concepts such as track, strategy and play. Intuitively, tracks are legal sequences of reachable states in a game that can be seen as descriptions of possible outcomes of the game.

Definition 4.2. A *track* is a finite sequence of states $\rho \in \mathrm{St}^*$ such that, for all $i \in [0, |\rho - 1|[$, if $(\rho)_i \in \mathrm{St}_0$ then there exists an action $a_0 \in \mathrm{Ac}_0$ such that $(\rho)_{i+1} = \mathtt{tr}((\rho)_i, a_0)$, else there exists an action $a_1 \in \mathrm{Ac}_1$ such that $(\rho)_{i+1} = \mathtt{tr}((\rho)_i, a_1)$, where $(\rho)_i$ denotes the i-st element of ρ. For a track ρ, by $last(\rho)$ we denote the last element of ρ and by $\rho_{\leq i}$ we denote the prefix track $(\rho)_0 \ldots (\rho)_i$. By $\mathrm{Trk} \subseteq \mathrm{St}^*$, we denote the set of tracks over St. By Trk_i we denote the set of tracks ρ in which $last(\rho) \in \mathrm{St}_i$. For simplicity, we assume that Trk contains only tracks starting at the initial state $s_I \in \mathrm{St}$.

A *strategy* represents a scheme for a player containing a precise choice of actions along an interaction with the other player. It is given as a function over tracks. The formal definition follows.

Definition 4.3. A *strategy* for $Player_i$ in a $2TRG$ G is a function $\sigma_i : \mathrm{Trk}_i \to \mathrm{Ac}_i$ that maps a track to an action.

The composition of strategies, one for each player in the game, induces a computation called *play*. More precisely, assume $Player_0$ and $Player_1$ take strategies σ_0 and σ_1, respectively. Their composition *induces* a play ρ such that $(\rho)_0 = s_I$ and for each $i \geq 0$ if $(\rho)_i \in \mathrm{St_o}$ then $(\rho)_{i+1} = \mathsf{tr}((\rho)_i, \sigma_0(\rho_{\leq i}))$, else $(\rho)_{i+1} = \mathsf{tr}((\rho)_i, \sigma_1(\rho_{\leq i}))$.

A strategy is winning for a player if all the plays induced by composing such a strategies with strategies from the adversarial player will enter a target state. If such a winning strategy exists we say that the player wins the game. Reachability games under perfect information are know to be *zero-sum*, *i.e.*, if $Player_0$ loses the game then $Player_1$ wins it and vice versa. The formal definition of *reachability winning condition* follows.

Definition 4.4. Let G be a $2TRG$ and $\mathrm{W} \subseteq \mathrm{St}$ a set of target states. $Player_0$ wins the game G, under the reachability condition, if he has a strategy such that for all strategies of $Player_1$ the resulting induced play will enter a state in W.

It is folklore that turn-based two-player reachability games are positional [44]. We recall that a strategy is positional if the moves of a player over a play only depends of the last state and a game is positional if positional strategies suffices to decide weather $Player_0$ wins the game. Directly from this result, the following corollary holds.

Corollary 4.5. Given a $2TRG$ G, a strategy σ_0 for $Player_0$, and a strategy σ_1 for $Player_1$, the induced play ρ is winning for $Player_0$ if there is $(\rho)_i \in \mathrm{W}$ with $0 \leq i \leq |\mathrm{St}| - 1$.

Hence, the corollary above just states that given a $2TRG$ game, $Player_0$ wins the game if he can reach a winning state in a number of steps bounded by the size of the set of states of the game.

Example 4.6. The two case studies that we have analyzed in Section 3 can be easily modeled using a $2TRG$. We now give some details. As set of states we use all the rooms in the maze, together with the status of their doors.

In the *Escape Game* the state $((0,0), \{d_1^c, d_3^c\})$ is the initial state, 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 $\mathrm{St} \subseteq \{((i,j), D_{i,j}) \mid 0 \leq i,j \leq 2\}$. The set of actions for the controller are $\mathrm{Ac}_{con} = \{open_{d_i} | \ 0 \leq i \leq 12\}$, *i.e.* he chooses a door to open. The set of actions for the human are $\mathrm{Ac}_{hum} = \{take_{d_i} | \ 0 \leq i \leq 12\}$, *i.e.* he chooses a door to take. Transitions are taken by the human in order to change the room (coordinates) or by the controller to change the status of 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. A possible track in which the human reaches the exit is $\rho = ((0,0), \{d_1^c, d_3^c\})((0,0), \{d_1^o, d_3^c\})((0,1), \{d_1^o, d_2^c, d_4^c\})((0,1), \{d_1^c, d_2^c, d_4^o\})((1,1), \{d_4^o, d_6^c, d_7^c, d_9^c\})$ $((1,1), \{d_4^c, d_6^c, d_7^o, d_9^c\})((1,2), \{d_5^c, d_7^o, d_{10}^c\})((1,2), \{d_5^c, d_7^c, d_{10}^o\})((2,2), \{d_{10}^o, d_{12}^c\})$.

In the same way, in *Cop and Robber Game* the initial state is $((1,1), \emptyset)$, where $\emptyset$ means that all doors are open. The set of actions for the cop are $\mathrm{Ac}_{cop} = \{block_{d_i} | \ 0 \leq i \leq 12\}$, *i.e.* he chooses a door to block. The set of actions for the robber are $\mathrm{Ac}_{rob} = \{take_{d_i} | \ 0 \leq i \leq 12\}$, *i.e.* he chooses a door to take.

5. Searching for Additional Winning Strategies

To check whether $Player_0$ has a winning strategy in a $2TRG$ G one can use a classic backward algorithm. We briefly recall it. Let $succ : \mathrm{St} \to 2^{\mathrm{St}}$ 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 $succ(s) \cap S \neq \emptyset$; or, (ii) $s \in \mathrm{St}_1$ and $succ(s) \subseteq S$. If S contains at a certain point the initial state, then $Player_0$ wins the game.

In case one wants to ensure that more than a winning strategy exists, the above algorithm becomes less appropriate. For this reason, we use instead a top-down automata-theoretic approach. To properly introduce this solution we first need to provide some auxiliary notation. Precisely, we introduce the concepts of *decision tree*, *strategy tree*, and *additional strategy tree*.

A decision tree simply collects all the tracks that come out from the interplays between the players. In other words, a decision tree can be seen as an unwinding of the game structure along with all possible combinations of players actions. The formal definition follows.

Definition 5.1. Given a $2TRG$ G, a *decision tree* is an St-labeled Ac-tree $<T, V>$, where ε is the root of T, $V(\varepsilon) = s_I$, and for all $v \in T$ we have that:

- if $last(v) \in \mathrm{Ac}_0$ then $last(anc(v)) \in \mathrm{Ac}_1$, otherwise $last(v) \in \mathrm{Ac}_1$;
- $V(v) = \mathsf{tr}(V(anc(v)), last(v))$.

We now introduce strategy trees that allow to collect, for each fixed strategy for $Player_i$, all possible responding strategies for $Player_{1-i}$, with $i \in \{0,1\}$. Therefore, the strategy tree is a tree where each node labeled with $s \in \mathrm{St}_i$ has an unique successor determined by the strategy for $Player_i$ and each node labeled with $s \in \mathrm{St}_{1-i}$ has $|\mathrm{Ac}_{1-i}|$ successors. Thus, a strategy tree is an opportune projection of the decision tree. The formal definition follows.

Definition 5.2. Given a $2TRG$ and a strategy σ for $Player_i$, a *strategy tree* for $Player_i$ is an St-labeled Ac-tree $<T, V>$, where ε is the root of T, $V(\varepsilon) = s_I$, and for all $v \in T$ we have that:

- if $last(v) \in \mathrm{Ac}_0$ then $last(anc(v)) \in \mathrm{Ac}_1$, otherwise $last(v) \in \mathrm{Ac}_1$;
- if $V(anc(v)) \in \mathrm{St}_i$ then $V(v) = \mathsf{tr}(V(anc(v)), \sigma(\rho))$, otherwise $V(v) = \mathsf{tr}(V(anc(v)), last(v))$;

where $\rho = (\rho)_0 \ldots (\rho)_{|v|-1}$ is a track from s_I, with $(\rho)_k = V(v_{\leq k})$ for each $0 \leq k \leq |v| - 1$.

Following the above definition and Definition 4.4, given a $2TRG$ G with a set of target states W, if G is determined then $Player_0$ wins the game and $Player_1$ loses it by simply checking the existence of a strategy tree for $Player_0$, that is a tree such that each path enters a state belonging to W. Such a tree is called a *winning-strategy tree* for $Player_0$.

In case we want to ensure that at least two winning strategies exist then, at a certain point along the tree, $Player_0$ must take two successors. We build a tree automaton that accepts exactly this kind of trees. We now give a definition of additional strategy trees and then we define the desired tree automata.

Definition 5.3. Given a $2TRG$ G and two strategies σ_1 and σ_2 for $Player_i$, an *additional strategy tree* for $Player_i$ is an St-labeled Ac-tree $<T, V>$ that satisfies the following properties:

- the root node is labeled with the initial state s_I of G;
- for each $x \in T$ that is not a leaf and it is labeled with state s of $Player_0$, 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 it is labeled with state s of $Player_1$, it holds that x has as children the set of $succ(s)$;
- each leaf of T corresponds to a target state in G;
- there exists at least one leaf in T that has an ancestor node x that corresponds to a $Player_0$ state in G and it has at least two children.

The above definition, but the last item, is the classical characterization of strategy tree. The last property further ensures that $Player_0$ has the ability to enforce at least two winning strategies no matter how $Player_1$ acts.

We now give the main result of this section, *i.e.* we show that it is possible to decide in linear time whether, in a 2*TRG*, $Player_0$ has more than a winning strategy. We later report on the application of this result along the case studies.

Theorem 5.4. For a 2*TRG* game G it is possible to decide in linear time whether $Player_0$ has more than a strategy to win the game.

Proof:
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 $Player_0$ over G. We describe the automaton. It uses $\mathrm{Q} = \mathrm{St} \times \{ok, split\}$ as set of states where ok and $split$ are flags and the latter is used to remember that along the tree $Player_0$ has to ensure the existence of two winning strategies by opportunely choosing a point where to "split". We set as alphabet $\Sigma = \mathrm{St}$ and initial state $q_0 = (s_I, split)$. For the transitions, starting from a state $q = (s, flag)$ and reading the symbol a, we have that:

$$\delta(q,a) = \begin{cases} (s', ok) & \text{if } s = a \text{ and } s \in \mathrm{St}_0 \text{ and } flag = ok; \\ ((s', ok) \wedge (s'', ok)) \vee (s', split) & \text{if } s = a \text{ and } s \in \mathrm{St}_0 \text{ and } flag = split; \\ (s_1, ok) \wedge \cdots \wedge (s_n, ok) & \text{if } s = a \text{ and } s \in \mathrm{St}_1 \text{ and } flag = ok; \\ (s_1, f_1) \wedge \cdots \wedge (s_n, f_n) & \text{if } s = a \text{ and } s \in \mathrm{St}_1 \text{ and } flag = split; \\ \emptyset & \text{otherwise.} \end{cases}$$

where $s', s'' \in succ(s)$ with $s' \neq s''$, $\{s_1, \ldots, s_n\} = succ(s)$, and $f_1, \ldots, f_n$ are flags in which there exists $1 \leq i \leq n$ such that $f_i = split$ and for all $j \neq i$, we have $f_j = ok$. Informally, given a state q, if q belongs to $Player_0$ and its flag is $split$ then there are one successor with flag $split$ or two successors with flag ok. Instead, if the flag is ok then there is only one successor with flag ok. In the case in which the state belongs to $Player_1$ and its flag is ok then there are n successors with flag ok. Finally, if the flag is $split$ then there are $n-1$ successors with flag ok and one successor with flag $split$.

The set of accepting states is $\mathrm{W} \times \{ok\}$. A tree is accepted by A if all the branches lead to a target state and there is a node labeled with a state in St_{o} that has at least two successors. By Corollary 4.5, we have that A considers only trees with depth until the number of states, so if no state in W is reached in $|\mathrm{St}|$ steps, then there is a loop over the states in the game model that forbids to reach states in W.

The size of the automaton is just linear in the size of the game. Moreover, by using the fact that, from [44], checking the emptiness of an NTA can be performed in linear time, the desired complexity result follows. □

Example 5.5. 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., $Player_0$ 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 is not empty. 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, by applying our solution technique, we end in an automaton that accepts an empty language. Conversely, by flipping the door d_4, the automaton accepts a tree that is witnessing of two different winning strategies each of them going through one of the two doors left unblocked by the cop.

For the sake of completeness, we report that in case of 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 reachability algorithm* in a way that we search for two paths leading to the target state. This can be done by just doubling the used logarithmic working space [49].

By means of the automata-theoretic approach, one can also check for other and more sophisticated "forms" of additional winning conditions. For example one can check whether $Player_0$ can win the game in case the opponent player is restricted to use *all but one* strategy. This check can be accomplished by first using a classic backward algorithm, introduced at the beginning of this section, for $Player_1$ and then the automaton introduced in the proof of Theorem 5.4, but used to collect all additional strategy trees for $Player_1$. Precisely, if the backward algorithm says that $Player_1$ wins the game and the automaton is empty, then the result holds. Indeed, the satisfaction of both these conditions says that $Player_1$ has one and only one strategy to beat all strategies of $Player_0$. Therefore, by removing this specific strategy, $Player_0$ wins the game. So, the explanation of the concept of *all but one* strategy derives.

Theorem 5.6. For a $2TRG$ game G it is possible to decide in linear time whether $Player_0$ can win G against all but one strategies of $Player_1$.

6. Games with Imperfect Information

In this section, we provide the setting of two-player turn-based finite games with imperfect information. As for the perfect information case, we consider here games along the reachability objective. The main difference with respect to the perfect case is that both players may not have full information about the moves performed by their opponents. Therefore, there could be cases in which a player has to come to a decision (which move to perform) without knowing exactly in which state he is. More precisely, we assume that the players act uniformly, so they use the same moves over states that are indistinguishable to them. The formal definition of these games follows.

Definition 6.1. A *turn-based two-player reachability game with imperfect information* ($2TRGI$, for short), played between $Player_0$ and $Player_1$, is a tuple $G \triangleq <\mathrm{St},\ s_I,\ \mathrm{Ac},\ \mathtt{tr},\ \mathrm{W},\ \cong_0,\cong_1>$, where St, s_I, Ac, $\mathtt{tr}$, and W are as in $2TRG$. Moreover, $\cong_0$ and $\cong_1$ are two equivalence relations over Ac.

Let $i \in \{0, 1\}$. The intuitive meaning of the equivalence relations is that two actions $a, a' \in \mathrm{Ac}_{1-i}$ such that $a \cong_i a'$ cannot be distinguished by $Player_i$. For this reason, we say that a and a' are *indistinguishable* to $Player_i$. By $[\mathrm{Ac}_i] \subseteq \mathrm{Ac}_i$ we denote the subset of actions that are distinguishable for $Player_{1-i}$. If two actions are indistinguishable then also the reached states are so [1]. A relation $\cong_i$ is said an *identity equivalence* if it holds that $a \cong_i a'$ iff $a = a'$. Note that, a $2TRGI$ has perfect information if the equivalence relations contain only identity relations.

To give the semantics of $2TRGI$s, we now introduce the concept of uniform strategy. A strategy is *uniform* if it adheres on the visibility of the players. To formally define it, we first give the notion of indistinguishability over tracks.

For a $Player_i$ and two tracks $\rho, \rho' \in \mathrm{Trk}$, we say that ρ and ρ' are indistinguishable to $Player_i$ iff $|\rho| = |\rho'| = m$ and for each $k \in \{0, \dots, m-1\}$ we have that $\overline{\mathtt{tr}}((\rho)_k, (\rho)_{k+1}) \cong_i \overline{\mathtt{tr}}((\rho')_k, (\rho')_{k+1})$, where $\overline{\mathtt{tr}}$ is the function that given two states s and s' returns the action a such that $s' = \mathtt{tr}(s, a)$. Note that $\overline{\mathtt{tr}}$ is well defined since it takes as input successive states coming from real tracks and it returns just one unique action due to the specific definition of $\mathtt{tr}$.

Definition 6.2. A strategy σ_i is *uniform* iff for every $\rho, \rho' \in \mathrm{Trk}$ that are indistinguishable for $Player_i$ we have that $\sigma(\rho) = \sigma(\rho')$.

Thus uniform strategies are based on observable actions. In the rest of the paper we only refer to uniform strategies. We continue by giving the definition of the the semantics of $2TRGI$, *i.e.* how $Player_0$ wins the game.

Definition 6.3. Let G be a $2TRGI$ and $\mathrm{W} \subseteq \mathrm{St}$ a set of target states. $Player_0$ wins the game G, under the reachability condition, if he has a *uniform strategy* such that for all uniform strategies of $Player_1$ the resulting induced play has at least one state in W.

Technically, a uniform strategy can be seen as an opportune mapping, over the decision tree, of a player's "strategy schema" built over the visibility part of the decision tree itself. In other words, the player first makes a decision over a set S of indistinguishable states and then this unique choice is used in the decision tree for each state in S. This makes the decision tree to become *uniform*. It is important to observe, however, that we use memoryfull strategies. This means that in a decision tree, the set S of indistinguishable states resides at the same level. To make this idea more precise, we now formalize the concept of *schema strategy tree* and *uniform strategy tree*.

Definition 6.4. Given a $2TRGI$ and a uniform strategy σ for $Player_i$, a *schema strategy tree* for $Player_i$ is a $\{\top, \bot\}$-labeled ($\mathrm{Ac}_i \cup [\mathrm{Ac}_{1-i}]$)-tree $<T, V>$, where ε is the root of T, $V(\varepsilon) = s_I$, and for all $v \in T$ we have that:

[1]For technical reasons, the indistinguishability over states follows from that one over actions. Thanks to this, the construction of the $2TRGI$ easily follows.

- if $last(v) \in \mathrm{Ac}_i$ then $last(anc(v)) \in [\mathrm{Ac}_{1-i}]$, otherwise $last(v) \in [\mathrm{Ac}_{1-i}]$;
- if $last(v) \in [\mathrm{Ac}_{1-i}]$ then $V(v) = \top$ else if $last(v) = \sigma(\rho)$ then $V(v) = \top$, otherwise $V(v) = \bot$;

where $\rho = (\rho)_0 \ldots (\rho)_{|v|-1}$ is a track from s_I, with $(\rho)_k = \mathsf{tr}((\rho)_{k-1}, last(v_{\leq k}))$ for each $0 \leq k \leq |v| - 1$.

Thus, in a schema strategy tree the $\top$ label indicates that $Player_i$ selects the corresponding set of visible states in the decision tree and the $\bot$ is used conversely [2]. In particular, the starting node of the game is the root of the schema strategy tree and it is always enabled; all nodes belonging to the adversarial player are always enabled; and one of the successors of $Player_i$ nodes is enabled in accordance with the uniform strategy σ. Straightforwardly, a *uniform strategy tree* is a projection of the decision tree along the schema strategy tree. In the next example we consider an extension of the Cop and Robber Game with imperfect information.

Example 6.5. Consider again the Cop and Robber Game example given in Section 3. Assume now, that we change the set of actions for the robber in $\mathrm{Ac}_{rob} = \{l, r, t, b\}$, where l, r, t, and b represent left, right, top, and bottom, respectively, and that the cop can always choose between two doors to enter, namely d_1 and d_2. Assume also that the cop can only recognize whether the robber moves horizontally or vertically. In other words, it holds that $l \cong_{cop} r$ and $t \cong_{cop} b$. Accordingly, the cop has only two uniform moves to perform, one for the first pair and one for the second. This is clearly different from the perfect information case where the cop has instead four moves to possibly catch the robber. More formally, in the imperfect information case, assuming that the robber moves first, we have four possible evolutions of the schema strategy tree. In all schema the root is labeled with $\top$ and it has two successors x and y, both labeled with $\top$ and corresponding to the actions l and r, and, t and b, possibly performed by the robber, respectively. Moreover, x and y have both two children. The four schema evolve by respectively placing to the children of x and y the following four combination of $\top$ and $\bot$: $((\top, \bot)(\top, \bot))$, $((\top, \bot)(\bot, \top))$, $((\bot, \top)(\top, \bot))$, and $((\bot, \top)(\bot, \top))$. For example, the second tuple corresponds to choose action d_1 in response to actions l and r and d_2 in response to t and b; similarly, the mining of the other tuple follows. Directly from this explanation it is no hard to build the corresponding uniform strategy trees.

7. Looking for Additional Winning Strategies in 2TRGI

In this section, we introduce an automaton-theoretic approach to solve $2TRGI$, taking as inspiration those introduced in [8, 32, 41, 44]. We start by analyzing the basic case of looking for a winning strategy. We recall that this problem is already investigated and solved in the case in which there is imperfect information over states [9, 50]. By these considerations, we show how to solve the case in which the imperfect information is over the actions. Subsequently, we extend the latter case to check whether the game also admits additional winning strategies.

Before starting we recall that positional strategies do not suffice to decide a game with imperfect information. Indeed, it is well known that $Player_0$ needs exponential memory *w.r.t.* the size of the states of the game in order to come up with a winning strategy in case it exists [50]. Therefore, we cannot use directly the approach exploited in Section 5. A possible direction to solve a game G with imperfect

[2] The use of $\top$ and $\bot$ is a classical solution in the automata theoretic approach to disable/enable successors, as it has been done in Module Checking [7] and the like.

information is to convert it, by means of a subset construction, in a game $\bar{G}$ with perfect information and solve it by using Theorem 5.4 (see for example [50]). With this translation one can individuate along the game $\bar{G}$ exponential strategies necessary to $Player_0$ for winning the game. As the subset construction involves an exponential blow-up and Theorem 5.4 provides a polynomial-time solution we get an overall exponential procedure. In this paper, however, we present a different and more elegant way to solve games with imperfect information. Precisely, we introduce a machinery that in polynomial time can represent exponential strategies. With more details, given a game with imperfect information G we construct an alternating tree automaton that accepts trees that represent uniform strategies under imperfect information. This is done by sending the same copy of the automaton (same direction) to all states that are indistinguishable to $Player_0$. Then, the automaton checks that in all these common directions $Player_0$ behaves the same and satisfies the reachability condition. Precisely, the automaton takes in input trees corresponding to $Player_0$'s strategies over the unwinding of the game by replacing nodes by the equivalence classes. The run instead is as usual, that is a $Player_0$ strategy over the total unwinding of the game. The beauty ot this approach resides on the fact that we do not make explicit the exponential strategies required to win the game but rather consider a polynomial compact representation of them by means of the automaton. Clearly, as the emptiness of alternating tree automata is exponential, we get the same overall exponential complexity as in the subset construction approach.

7.1. Solution 2TRGI

To solve $2TRGI$, we use an automata-approach via alternating tree automata. The idea is to read a $\{\top, \bot\}$-labeled $(\mathrm{Ac}_0 \cup [\mathrm{Ac}_1])$-tree such that more copies of the automaton are sent to the same directions along the class of equivalence over $[\mathrm{Ac}_1]$.

Theorem 7.1. Given a $2TRGI$ G played by $Player_0$ and $Player_1$, the problem of deciding whether $Player_0$ wins the game is EXPTIME-COMPLETE.

Proof:
Let G be a $2TRGI$. For the lower bound, we recall that deciding the winner in a 2-player turn-based games with imperfect information is EXPTIME-HARD [9, 50].

For the upper bound, we use an automata-theoretic approach. Precisely, we build an ATA A that accepts all schema strategy trees for $Player_0$ over G. The automaton, therefore will send more copies on the same direction of the input tree when they correspond to hidden actions. Then it will check the consistency with the states on the fly by taking in consideration the information stored in the node of the tree. This can be simply checked by means of a binary counter along with the states of the automaton. For the sake of readability we omit this.

The automaton uses as set of states $Q = \mathrm{St} \times \mathrm{St} \times \{\top, \bot\} \times \{0, 1\}$ and alphabet $\Sigma = \{\top, \bot\}$. Note that, we use in Q a duplication of game states as we want to remember the game state associated to the parent node while traversing the tree. For the initial state we set $q_0 = (s_I, s_I, \top, 0)$, *i.e.*, for simplicity the parent game state associated to the root of the tree is the game state itself. The flag $f \in \{0, 1\}$ indicates whether along a path we have entered a target state, in that case we move f from 0 to 1. Given a state $q = (s, s', t, f)$, the transition relation is defined as:

$$\delta(q,t') = \begin{cases} \bigwedge_{a_0 \in \mathrm{Ac}_0}(d,(s',s'',\top,f')) & \text{if } s' \in \mathrm{St}_0 \text{ and } t'=\top \text{ and } t=\top; \\ \bigwedge_{a_1 \in \mathrm{Ac}_1}(d,(s',s'',\top,f')) & \text{if } s' \in \mathrm{St}_1 \text{ and } t'=\top \text{ and } t=\top; \\ false & \text{if } t'=\top \text{ and } t=\bot; \\ true & \text{if } t'=\bot. \end{cases}$$

where if $s' \in \mathrm{St}_0$ then $s'' = tr(s', a_0)$ and d is in accordance with $|\mathrm{Ac}_1|$, else $s'' = tr(s', a_1)$ and d is in accordance with $|\mathrm{Ac}_0|$; if $q' \in \mathrm{W}$ then $f' = 1$ otherwise $f' = f$. Informally, given a state q, if q belongs to $Player_0$ (*resp.*, $Player_1$) and it is enabled then there are $|\mathrm{Ac}_0|$ (*resp.*, $|\mathrm{Ac}_1|$) enabled successors. Instead, if q is disabled then the automaton returns $false$. Finally, if the automaton reads the symbol $\bot$ then it returns $true$.

The set of accepted states is $\mathrm{F} = \{(s, s', t, f) : s, s' \in \mathrm{St} \wedge t = \top \wedge f = 1\}$. Recall that an input tree is accepted if there exists a run whose leaves are all labeled with accepting states. In our setting this means that an input tree simulates a schema strategy tree for $Player_0$. So, if the automaton is not empty then $Player_0$ wins the game, *i.e.*, there exists a uniform strategy for him.

The required computational complexity of the solution follows by considering that: *(i)* the size of the automaton is polynomial in the size of the game, *(ii)* to check its emptiness can be performed in exponential time [51, 47]. □

7.2. Additional Winning Strategies for 2TRGI

In this section we describe the main result of this work, *i.e.*, we show an elementary solution to ensure that more than a winning strategy exists in $2TRGI$s. As we have anticipated earlier we use an opportune extension of the automata-theoretic approach we have introduced in the previous sections.

First of all, we formalize the concept of *schema additional strategy tree*.

Definition 7.2. Given a $2TRGI$ and two uniform strategies σ and σ' for $Player_i$, a *schema additional strategy tree* for $Player_i$ is a $\{\top, \bot\}$-labeled $(\mathrm{Ac}_i \cup [\mathrm{Ac}_{1-i}])$-tree $< T, V >$, where ε is the root of T, $V(\varepsilon) = s_I$, and for all $v \in T$ we have that:

- if $last(v) \in \mathrm{Ac}_i$ then $last(anc(v)) \in [\mathrm{Ac}_{1-i}]$, otherwise $last(v) \in [\mathrm{Ac}_{1-i}]$;
- if $last(v) \in [\mathrm{Ac}_{1-i}]$ then $V(v) = \top$ else if $last(v) = \sigma(\rho)$ or $last(v) = \sigma'(\rho)$ then $V(v) = \top$, otherwise $V(v) = \bot$;

where $\rho = (\rho)_0 \ldots (\rho)_{|v|-1}$ is a track from s_I, with $(\rho)_k = \mathsf{tr}((\rho)_{k-1}, last(v_{\leq k}))$ for each $0 \leq k \leq |v| - 1$.

Informally, a schema additional strategy tree is a schema strategy tree in which at a certain point along the tree, a state of $Player_0$ has to two successors. We build a tree automaton that accepts exactly this kind of trees. Now, we have all ingredients to give the following result.

Theorem 7.3. Given a $2TRGI$ G played by $Player_0$ and $Player_1$, the problem of deciding whether $Player_0$ has more than a uniform strategy to win the game is in EXPTIME-COMPLETE.

Proof:
Let G be a $2TRGI$. For the lower bound, we recall that deciding the winner in a 2-player turn-based games with imperfect information is EXPTIME-HARD [9, 50].

For the upper bound, we use an automata-theoretic approach. Precisely, we build an ATA A that accepts all schema additional strategy trees for $Player_0$ over G. Since the automaton sends more copies on the same direction of the input tree when they correspond to hidden actions, then it checks the consistency with the states on the fly by taking in consideration the information stored in the node of the tree. In detail, the automaton uses as set of states $Q = \mathrm{St} \times \mathrm{St} \times \{\top, \bot\} \times \{0, 1\} \times \{ok, split\}$, where given a state $q = (s, s', t, f, \bar{f})$ we have that s is the parent of s', s' is the actual state, t is used to disable/enable the state, f is a flag indicating whether along the path we have entered in a target state, and $\bar{f}$ is a flag indicating whether along the path there was a state of $Player_0$ with two successors. The alphabet is $\Sigma = \{\top, \bot\}$ and the initial state is $q_0 = (s_I, s_I, \top, 0, split)$. Given a state $q = (s, s', t, f, \bar{f})$, the transition relation is defined as follows:

$$\delta(q, t') = \begin{cases} \bigwedge_{a_0 \in \mathrm{Ac}_0} (d, (s', s'', \top, f', ok)) & \text{if } s' \in \mathrm{St}_0 \text{ and } t' = \top \text{ and } t = \top \text{ and } \bar{f} = ok; \\ \bigwedge_{a_0 \in \mathrm{Ac}_0} \bigvee_{\bar{f}' \in \{ok, split\}} (d, (s', s'', \top, f', \bar{f}')) & \text{if } s' \in \mathrm{St}_0 \text{ and } t' = \top \text{ and } t = \top \text{ and } \bar{f} = split; \\ \bigwedge_{a_1 \in \mathrm{Ac}_1} (d, (s', s'', \top, f', ok)) & \text{if } s' \in \mathrm{St}_1 \text{ and } t' = \top \text{ and } t = \top \text{ and } \bar{f} = ok; \\ \bigwedge_{a_1 \in \mathrm{Ac}_1} \bigvee_{\bar{f}' \in \{ok, split\}} (d, (s', s'', \top, f', \bar{f}')) & \text{if } s' \in \mathrm{St}_1 \text{ and } t' = \top \text{ and } t = \top \text{ and } \bar{f} = split; \\ false & \text{if } t' = \top \text{ and } t = \bot; \\ true & \text{if } t' = \bot. \end{cases}$$

where it holds that if $s' \in \mathrm{St}_0$ then $s'' = tr(s', a_0)$ and d is in accordance with $|\mathrm{Ac}_1|$, otherwise $s'' = tr(s', a_1)$ and d is in accordance with $|\mathrm{Ac}_0|$; if $q' \in \mathrm{W}$ then $f' = 1$ otherwise $f' = f$. Informally, given a state q, if q belongs to $Player_0$, it is enabled, and its flag is $split$ so there are $|\mathrm{Ac}_0|$ enabled successors, such that it holds that either all of them have $split$ has flag, or at least two of them have ok as flag . Instead, if the state q is enabled and its flag is ok then there are $|\mathrm{Ac}_0|$ enabled successors and all of them have the flag ok. If the state q belongs to $Player_1$, it is enabled, and its flag is ok, thus there are $|\mathrm{Ac}_1|$ enabled successors such that all of them have the flag ok. Instead, if the state is enabled and its flag is $split$ then there are $|\mathrm{Ac}_1|$ enabled successors such that at least one successor has flag $split$. In the case in which the state q is disabled then the automaton returns $false$. Finally, if the automaton reads the symbol $\bot$ then it returns $true$.

The set of accepted states is $\mathrm{F} = \{(s, s', t, f, \bar{f}) : s, s' \in \mathrm{St} \wedge t = \top \wedge f = 1 \wedge \bar{f} = ok\}$. Recall that an input tree is accepted if there exists a run whose leaves are all labeled with accepting states. In our setting this means that an input tree simulates a schema additional strategy tree for $Player_0$. So, if the automaton is not empty then $Player_0$ wins the game, *i.e.*, there exists a schema additional strategy tree for him. The required computational complexity of the solution follows by considering that: *(i)* the size of the automaton is polynomial in the size of the game, *(ii)* to check its emptiness can be performed in exponential time [51, 47]. □

Finally, also in the imperfect information case one can repeat the same reasoning done in Section 5 about "all but one" strategies. Indeed, it is sufficient to use the automata in the proofs of Theorem 7.1 and

Theorem 7.3 from the viewpoint of $Player_1$. Indeed, the result follows by checking whether the former automaton is not empty and the latter automaton is empty. Consequently, the following result holds.

Theorem 7.4. For a $2TRGI$ game G it is possible to decide in EXPTIME-COMPLETE whether $Player_0$ has a uniform strategy against all but one uniform strategies of $Player_1$.

8. 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. Our approach works with optimal asymptotic complexity both in the case the players have perfect information about the moves performed by their adversarial or not. Overall, this is the first work dealing with the counting of strategies in the imperfect information setting we are aware of.

We have showed how our approach 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. For example, it can be used to solve the *Unique Nash Equilibrium* problem, in an extensive game form.

This work opens to several interesting questions and extensions. An interesting direction is to consider the counting of strategies in multi-agent concurrent games. This kind of games have several interesting applications in artificial intelligence [6]. Some works along this line have been done, but not for finite games, nor in the imperfect information setting. As another direction of work, one can consider some kind of hybrid game, where one can opportunely combine teams of players working concurrently with some others playing in a turn-based manner as in [52, 53, 54]. 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 [55, 56].

Acknowledgments

We thank prof. Alberto Pettorossi for useful discussion regarding the complexity of the problem of checking whether more than a winning strategy exists, in case of one-player games. We also thank the anonymous referee for helpful comments.

References

[1] Malvone V, Murano A, Sorrentino L. Games with Additional Winning Strategies. In: CILC 2015; 2015. p. 175–180.

[2] Harel D, Pnueli A. On the Development of Reactive Systems. Springer; 1985.

[3] Myerson RB. Game Theory: Analysis of Conflict. Harvard University Press; 1991.

[4] Smith JM. Evolution and the Theory of Games. Cambridge university press; 1982.

[5] Alur R, Henzinger TA, Kupferman O. Alternating-Time Temporal Logic. Journal of the ACM. 2002;49(5):672–713.

[6] Wooldridge M. An Introduction to Multi Agent Systems. John Wiley & Sons; 2002.

[7] Kupferman O, Vardi MY, Wolper P. Module Checking. Information and Computation. 2001;164(2):322–344.

[8] Kupferman O, Vardi MY. Module Checking Revisited. In: Computer Aided Verification'97. LNCS 1254. Springer; 1997. p. 36–47.

[9] Reif JH. The Complexity of Two-Player Games of Incomplete Information. J Comput Syst Sci. 1984;29(2):274–301.

[10] Pnueli A, Rosner R. On the Synthesis of a Reactive Module. In: Principles of Programming Languages'89. Association for Computing Machinery; 1989. p. 179–190.

[11] Malvone V, Murano A, Sorrentino L. Hiding Actions in Concurrent Games. In: ECAI 2016. vol. 285 of FAIA; 2016. p. 1686–1687.

[12] Malvone V, Murano A, Sorrentino L. Hiding Actions in Multi-Player Games. In: AAMAS 2017; 2017. p. 1205–1213.

[13] Altman E, Kameda H, Hosokawa Y. Nash Equilibria in Load Balancing in Distributed Computer Systems. IGTR. 2002;4(2):91–100.

[14] Papavassilopoulos GP, Cruz JB. On the Uniqueness of Nash Strategies for a Class of Analytic Differential Games. Journal of Optimization Theory and Applications. 1979;27(2):309–314.

[15] Cornes R, Hartley R, Sandler T. An Elementary Proof via Contraction. Journal of Public Economic Theory. 1999;1(4):499–509.

[16] Orda A, Rom R, Shimkin N. Competitive Routing in Multiuser Communication Networks. IEEE/ACM Trans Netw. 1993;1(5):510–521.

[17] Bergstrom TC, Blume LE, Varian HR. On the Private Provision of Public Goods. Journal of Public Economics. 1986;29(1):25–49.

[18] Fraser CD. The Uniqueness of Nash Equilibrium in the Private Provision of Public Goods: an Alternative Proof. Journal of Public Economics. 1992;49(3):389–390.

[19] Glazer A, Konrad KA. Private Provision of Public Goods, Limited Tax Deducibility, and Crowding Out. FinanzArchiv / Public Finance Analysis. 1993;50(2):203–216.

[20] Malvone V, Mogavero F, Murano A, Sorrentino L. On the Counting of Strategies. In: TIME 2015; 2015. p. 170–179.

[21] Aminof B, Malvone V, Murano A, Rubin S. Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria. In: AAMAS 2016; 2016. p. 698–706.

[22] D Simchi-Levi JB X Chen. The Logic of Logistics: Theory, Algorithms, and Applications for Logistics Management. Science and Business Media. Springer; 2013.

[23] Zhang Y, Guizani M. Game Theory for Wireless Communications and Networking. CRC Press; 2011.

[24] Pavel L. Game Theory for Control of Optical Networks. Science and Business Media. Springer; 2012.

[25] Kitano H, Tadokoro S, Noda I, Matsubara H, Takahashi T, Shinjou A, et al. Robocup rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research. In: IEEE. vol. 6; 1999. p. 739–743.

[26] Kitano H, Tadokoro S. Robocup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. AI magazine. 2001;22(1):39.

[27] Chang M, Tseng Y, Chen J. A Scenario Planning Approach for the Flood Emergency Logistics Preparation Problem under Uncertainty. Transportation Research Part E: Logistics and Transportation Review. 2007;43(6):737–754.

[28] Alur R, Courcoubetis C, Yannakakis M. Distinguishing Tests for Nondeterministic and Probabilistic Machines. In: ACM; 1995. p. 363–372.

[29] Berthon R, Maubert B, Murano A, Rubin S, Vardi MY. Strategy logic with imperfect information. In: LICS; 2017. p. 1–12.

[30] Belardinelli F, Lomuscio A, Murano A, Rubin S. Verification of Broadcasting Multi-Agent Systems against an Epistemic Strategy Logic. In: IJCAI 2017; 2017. p. 91–97.

[31] Bonatti PA, Lutz C, Murano A, Vardi MY. The Complexity of Enriched muCalculi. Logical Methods in Computer Science. 2008;4(3):1–27.

[32] Ferrante A, Murano A, Parente M. Enriched Mu-Calculi Module Checking. Logical Methods in Computer Science. 2008;4(3):1–21.

[33] Fine K. In So Many Possible Worlds. Notre Dame Journal of Formal Logic. 1972;13:516–520.

[34] Grädel E, Otto M, Rosen E. Two-Variable Logic with Counting is Decidable. In: Logic in Computer Science'97. IEEE Computer Society; 1997. p. 306–317.

[35] Horrocks I, Sattler U. Decidability of SHIQ with Complex Role Inclusion Axioms. Artif Intell. 2004;160(1-2):79–104.

[36] Calvanese D, Giacomo GD, Lenzerini M, Vardi MY. Node Selection Query Languages for Trees. In: AAAI; 2010. p. 279–284.

[37] Calvanese D, Eiter T, Ortiz M. Answering Regular Path Queries in Expressive Description Logics via Alternating Tree-Automata. Inf Comput. 2014;237:12–55.

[38] Baader F, Borgwardt S, Lippmann M. Temporal Conjunctive Queries in Expressive Description Logics with Transitive Roles. In: AI 2015; 2015. p. 21–33.

[39] Feier C, Eiter T. Reasoning with Forest Logic Programs Using Fully Enriched Automata. In: LPNMR 2015; 2015. p. 346–353.

[40] Kupferman O, Sattler U, Vardi MY. The Complexity of the Graded muCalculus. In: Conference on Automated Deduction'02. LNCS 2392. Springer; 2002. p. 423–437.

[41] Bianco A, Mogavero F, Murano A. Graded Computation Tree Logic. Transactions On Computational Logic. 2012;13(3):25:1–53.

[42] Aminof B, Murano A, Rubin S. On CTL* with Graded Path Modalities. In: LPAR-20; 2015. p. 281–296.

[43] Bernet J, Janin D, Walukiewicz I. Permissive strategies: from parity games to safety games. RAIRO-Theoretical Informatics and Applications. 2002;36(3):261–275.

[44] Thomas W. Infinite Trees and Automaton Definable Relations over Omega-Words. In: STACS'90; 1990. p. 263–277.

[45] Kupferman O, Vardi MY. Synthesis with incomplete informatio. In: Advances in Temporal Logic. Springer; 2000. p. 109–127.

[46] Vardi MY, Wolper P. Automata-Theoretic Techniques for Modal Logics of Programs. Journal of Computer and System Science. 1986;32(2):183–221.

[47] Kupferman O, Vardi MY, Wolper P. An Automata Theoretic Approach to Branching-Time Model Checking. Journal of the ACM. 2000;47(2):312–360.

[48] Emerson EA, Jutla CS. Tree Automata, muCalculus, and Determinacy. In: Foundation of Computer Science'91. IEEE Computer Society; 1991. p. 368–377.

[49] Sipser M. Introduction to the Theory of Computation. vol. 2. Thomson Course Technology Boston; 2006.

[50] Chatterjee K, Doyen L, Henzinger TA, Raskin JF. Algorithms for omega-regular games with imperfect information. Logical Methods in Computer Science. 2007;3(4):1–23.

[51] Emerson EA, Jutla CS. The Complexity of Tree Automata and Logics of Programs (Extended Abstract). In: Foundation of Computer Science'88. IEEE Computer Society; 1988. p. 328–337.

[52] Jamroga W, Murano A. On Module Checking and Strategies. In: Autonomous Agents and MultiAgent Systems'14. International Foundation for Autonomous Agents and Multiagent Systems; 2014. p. 701–708.

[53] Jamroga W, Murano A. Module Checking of Strategic Ability. In: Autonomous Agents and MultiAgent Systems'15. International Foundation for Autonomous Agents and Multiagent Systems; 2015. p. 227–235.

[54] Murano A, Sorrentino L. A Game-Based Model for Human-Robots Interaction. In: WOA'15. vol. 1382 of CEUR Workshop Proceedings. CEUR-WS.org; 2015. p. 146–150.

[55] Bozzelli L, Murano A, Peron A. Pushdown Module Checking. Formal Methods in System Design. 2010;36(1):65–95.

[56] Murano A, Perelli G. Pushdown Multi-Agent System Verification. In: IJCAI 2015; 2015. p. 1090–1097.