The Complexity of LTL Rational Synthesis

Orna Kupferman and Noam Shenwald

School of Computer Science and Engineering, The Hebrew University, Israel

Abstract. In *rational synthesis*, we automatically construct a reactive system that satisfies its specification in all rational environments, namely environments that have objectives and act to fulfill them. We complete the study of the complexity of LTL rational synthesis. Our contribution is threefold. First, we tighten the known upper bounds for settings that were left open in earlier work. Second, our complexity analysis is parametric, and we describe tight upper and lower bounds in each of the problem parameters: the game graph, the objectives of the system components, and the objectives of the environment components. Third, we generalize the definition of rational synthesis, combining the cooperative and non-cooperative approaches studied in earlier work, and extend our complexity analysis to the general definition.

1 Introduction

Synthesis is the automated construction of a system from its specification. The basic idea is simple and appealing: instead of developing a system and verifying that it adheres to its specification, we use an automated procedure that, given a specification, constructs a system that is correct by construction, thus enabling the designers to focus on *what* the system should do rather than *how* to do it. A reactive system interacts with its environment and should satisfy its specification in all environments [9, 25]. Accordingly, synthesis corresponds to a *zero-sum game* between the system and the environment, where they together generate a computation, the system wins if the computation satisfies the specification, and otherwise, the environment wins.

In practice, the requirement to satisfy the specification in all environments is often too strong. Therefore, it is common to add assumptions on the environment. An assumption may be direct, say a specification that restricts the possible behaviors of the environment [6], or less direct, say a bound on the size of the environment or other resources it uses [14]. In [12], the authors suggest a conceptual assumption on the environment, namely its rationality: *Rational synthesis* is based on the idea that the components composing the environment typically have objectives of their own, and they act to achieve their objectives. For example, clients interacting with a server typically have objectives other than to fail the server. As shown in [12], the system can capitalize on the rationality and objectives of components that compose its environment. Adding rationality into the picture makes the corresponding game *non-zero-sum* [22], thus objectives of different players may overlap.

The interesting questions about non-zero-sum games concern *stable outcomes*, in particular *Nash equilibria* (NE) [21]. More formally, each of the players in the game has a *strategy* that directs her which actions to take; a *profile* is a vector of strategies, one for each player; each profile has an *outcome* (in our case, the computation generated when the system and the environment follow their strategies); and a profile is an NE if no player has an incentive to deviate from it (in our case, to change her strategy in a way that would cause the outcome of the new profile to satisfy her objective).

Two approaches to rational synthesis have been studied. In *cooperative* rational synthesis (CRS) [12], the desired output is an NE profile whose outcome satisfies the objective of the system. Thus, in CRS, we assume that we can suggest strategies to the environment players, and once they have no incentive to deviate from these strategies, they follow them. Then, in *non-cooperative* rational synthesis (NRS) [15], the desired output is a strategy for the system player such that the objective of the system is satisfied in the outcome of all NE profiles that include this strategy. Thus, in NRS, the environment players are rational, but we cannot suggest them a strategy.

The cooperative and non-cooperative approaches correspond to different settings in reality, having to do both with the technical ability to communicate a strategy to the environment players, say due to different architectures, as well as the willingness of the environment players to follow a suggested strategy. As shown in [1], the two approaches are related to the two stability-inefficiency measures of *price of stability* [3] and *price of anarchy* [16, 23]. Additional related work includes *rational verification* [27, **?**], where we check that a given system satisfies its specification when interacting with a rational environment, and extensions of rational synthesis to richer settings (multi-valued, partial visibility, and more) [4, 13, 18].

The *complexity* of rational synthesis was first studied for the case the input to the problem is the objectives of the players, given by LTL formulas. In this setting, CRS is in 2EXPTIME [12], whereas the best known upper bound for NRS until recently was 3EXPTIME [15] (the paper specifies a 2EXPTIME upper bound, but a careful analysis of the algorithm reveals that it is actually in 3EXPTIME), improved to 2EXPTIME for *turn-based* games with two players [18]. The complexity analysis above suggests that rational synthesis is not harder than traditional synthesis. One may wonder whether this has to do with the doubly-exponential translation of LTL to deterministic automata, which dominates the complexity. To answer this question, [10] studies the complexity of rational synthesis where the objectives of the players are given by ω-regular winning conditions in a game graph (e.g., reachability, Büchi, and parity). The analysis in [10] also distinguishes between the case the number of players is fixed and the case it is not. As shown there, in most cases the complexity of the rational variant coincides with the complexity of the zero-sum game. In some cases, however, it does not. For example, while the problem of deciding Rabin games is NP-complete [11], the best algorithm for solving CRS with Rabin objectives

is in P^{NP}, going up to PSPACE-complete in NRS, and going higher when the number of players is not fixed [10].

In this work, we complete the study of the complexity of LTL rational synthesis. Our contribution is threefold. First, we tighten the known upper bound for NRS for settings with three or more players and for *concurrent* games, which were left open in [10, 18]. Second, our complexity analysis is *parametric*, and we describe tight upper and lower bounds in each of the problem parameters: the game graph, the objectives of the system players, and the objectives of the environment players. Third, we generalize the definition of rational synthesis, combining the cooperative and non-cooperative approaches, and extend our complexity analysis to the general definition. Below we elaborate on each of the contributions.

Let us start with the generalization of the problem. In our general definition, we may suggest a strategy only to a subset of the environment players. Thus, we distinguish between three types of players: controllable, cooperative uncontrollable, and non-cooperative uncontrollable. Then, in the (general) rational-synthesis (RS) problem, we are given a labeled graph and LTL formulas that specify the objectives of the players, and we seek strategies for the controllable and the cooperative-uncontrollable players such that the objectives of the controllable players are satisfied in the outcome of every NE profile that extends these strategies. Note that CRS and NRS can be viewed as special cases of RS where the uncontrollable players are all cooperative or all non-cooperative.

In the tight-complexity front, our algorithms reduce rational synthesis to the nonemptiness problem of tree automata. The automata accept *certified strategy trees*: trees that are labeled by both a strategy for the controllable player[1] and information about uncontrollable players that deviate and the strategies to which they deviate. The most technically-challenging algorithm we describe is for NRS in the concurrent setting. While in the turn-based setting, we need a single player that deviates in order to justify a path in which the objective of the controllable player is not satisfied, in the concurrent setting, where the players choose actions simultaneously and independently, we need to consider sets of uncontrollable players. This makes the certificate much more complex. In particular, it involves labels from an exponential alphabet, which introduces an additional challenge, namely a need to decompose labels along branches in the tree. Also, while in the turn-based setting, an NE always exists, concurrent games with three or more players need not have an NE [8], and so a certified strategy tree should also certify the existence of an NE.

Finally, in the parameterized-complexity front, the fact our algorithms use tree automata (rather than a translation to Strategy Logic [7], which has been the case in [12, 15]), enables us to analyze the complexity in each of the parameters of the problem: the game graph G, the objective ψ_1 of the controllable player, and the objectives $\psi_2, \ldots, \psi_k$ of the uncontrollable players. For CRS, [18] studies the

[1] It is easy to see that several controllable components can be merged to a single one.

parameterized complexity in turn-based games with two players.[2] The algorithm there is based on a distinction between the case the uncontrollable player satisfies her objective and the case she does not. Generalizing this to an arbitrary number of players, we parameterize solutions with the set of the uncontrollable players whose objectives are satisfied, and give a uniform solution to all cases. This also enables us to seek solutions that favor some or all uncontrollable players.

We show that the complexity of CRS is polynomial in $|G|$, doubly-exponential in $|\psi_2|, \ldots, |\psi_k|$, and only exponential in $|\psi_1|$. Thus, in terms of the system specification, CRS is in fact easier than traditional synthesis! Once we move to NRS or RS, the complexity becomes doubly exponential in all objectives. We describe tight lower bounds for the different parameters, and we show that they are valid already for the case $k = 2$ and the game is turn based. Specifically, we prove that CRS is EXPTIME-hard even when G and ψ_2 are fixed, and is 2EXPTIME-hard even when G and ψ_1 are fixed. Similarly, NRS is 2EXPTIME-hard even when only one of ψ_1 and ψ_2 is not fixed. In order to see the technical challenge in our lower-bound proofs, consider the current 2EXPTIME lower-bound proof for CRS, where synthesis of an objective ψ for the system is reduced to CRS with objectives ψ for the system and $\neg\psi$ for the environment. The reduction crucially depends on both objectives not being fixed, and just changing either of them to *True* or *False* does not do the trick. In order to get 2EXPTIME-hardness in $|\psi_2|$, we need to cleverly manipulate both G and ψ_1.

Together, our results complete the complexity picture for a generalized definition of rational synthesis, for both turn-based and concurrent systems, with any number of components, and with the exact dependencies in each of the parameters of the problem.

2 Preliminaries

2.1 LTL, trees, and automata

The logic LTL is used for specifying on-going behaviors of reactive systems [24]. Formulas of LTL are constructed from a set AP of atomic propositions using the usual Boolean operators and the temporal operators G ("always") and F ("eventually"), X ("next time") and U ("until"). The semantics of LTL is defined with respect to infinite computations in $(2^{AP})^\omega$. We are going to use LTL for specifying the objectives of the system and the components composing the environment.

Given a set D of directions, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot d \in T$, where $x \in D^*$ and $d \in D$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $x \in T$, the nodes $x \cdot d$, for $d \in D$, are the *successors* of x, and the *direction* of node $x \cdot d$ is d. A *path* h in a tree T is a set $h \subseteq T$ such that $\varepsilon \in h$ and for every $x \in h$, either x is a leaf or

[2] The study in [18] considers *perspective games* [19], which adds the challenge of *partial visibility* on top of rational synthesis, but the results there imply the desired bounds for the case of full visibility.

there exists a unique $d \in D$ such that $x \cdot d \in h$. We sometimes refer to paths in T as words in D^* or D^ω. For a finite path $h \subseteq D^*$ and a finite or infinite path $h' \subseteq D^*$, we use $h \preceq h'$ to indicate that h is a prefix of h', thus $h \subseteq h'$. Given an alphabet Σ, a *Σ-labeled D-tree* is a pair $\langle T, \tau \rangle$ where T is a tree and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ.

Our algorithms use *automata on infinite words and trees*. An automaton $\mathcal{A}$ on infinite words over an alphabet Σ defines a language $L(\mathcal{A}) \subseteq \Sigma^\omega$. An automaton $\mathcal{A}$ on Σ-labeled D-trees defines a language of such trees. Details can be found in Appendix A. The *size* of $\mathcal{A}$, denoted $|\mathcal{A}|$, is the sum of lengths of the description of its transition function. We denote different types of automata by three-letter acronyms in $\{D, N, U, A\} \times \{F, B, C, P\} \times \{W, T\}$, where the first letter describes the branching mode of the automaton (deterministic, nondeterministic, universal, or alternating), the second letter describes the acceptance condition (finite, Büchi, co-Büchi, or parity), and the third letter describes the object over which the automaton runs (words or trees). For example, UCT stands for a universal co-Büchi tree automaton.

2.2 Concurrent multiplayer games

For $k \geq 1$, let $[k] = \{1, \ldots, k\}$. A *k-player game graph* is a tuple $G = \langle AP, V, v_0, \{A_i\}_{i \in [k]}, \{\kappa_i\}_{i \in [k]}, \delta, \tau \rangle$, where AP is a set of atomic propositions, V is a set of vertices, $v_0 \in V$ is an initial vertex, and for $i \in [k]$, the set A_i is a set of actions of Player i, and $\kappa_i : V \to 2^{A_i}$ specifies the set of actions that Player i can take at each vertex.

A *move* in G is a tuple $\langle a_1, \ldots, a_k \rangle \in A_1 \times \cdots \times A_k$, describing possible choices of actions for all k players. A move $\langle a_1, \ldots, a_k \rangle$ is *possible* for vertex $v \in V$ if $a_i \in \kappa_i(v)$ for all $i \in [k]$. Then, the transition function $\delta : V \times A_1 \times \cdots \times A_k \to V$ is a deterministic function that maps each vertex and possible move for it to a successor vertex. Finally, the function $\tau : V \to 2^{AP}$ maps each vertex to the set of atomic propositions that hold in it.

A *game* is a tuple $\mathcal{G} = \langle G, \{\psi_i\}_{i \in [k]} \rangle$, where G is a k-player game graph, and ψ_i, for $i \in [k]$, is an LTL formula over AP, describing the *objective* of Player i. In a beginning of a play in the game, a token is placed on v_0. Then, at each round, the players choose actions simultaneously and independently of the other players, and the induced move determines the successor vertex. Repeating this, the players generate a *play* $\rho = v_0, v_1, \ldots$ in G, which induces the *computation* $\tau(\rho) = \tau(v_0), \tau(v_1), \ldots \in (2^{AP})^\omega$. For every $i \in [k]$, Player i aims for a play whose computation satisfies ψ_i. For an LTL formula ψ, let $L(\psi) \subseteq (2^{AP})^\omega$ be the set of computations that satisfy ψ.

A *strategy* for Player i is a function $f_i : V^+ \to A_i$ that maps histories of the game to an action suggested to Player i. The suggestion has to be consistent with κ_i. Thus, for every $v_0 v_1 \cdots v_j \in V^+$, we have that $f_i(v_0 v_1 \cdots v_j) \in \kappa_i(v_j)$. A *profile* is a tuple $\pi = \langle f_1, \ldots, f_k \rangle$ of strategies, one for each player. The *outcome* of a profile $\pi = \langle f_1, \ldots, f_k \rangle$ is the play obtained when the players follow their strategies. Formally, $\mathsf{Outcome}(\pi) = v_0, v_1, \ldots$ is such that for all $j \geq 0$, we have that $v_{j+1} = \delta(v_j, \langle f_1(v_0 \cdots v_j), \cdots, f_k(v_0 \cdots v_j) \rangle)$. For a subset $S \subseteq [k]$ of

players, an *S-profile* is a set of strategies, one for each player in S. We say that a profile π *extends* an S-profile π' if the players in S use in π their strategies in π'.

Consider a profile π. The set of *winners* in π, denoted $\mathsf{Win}(\pi)$, is the set of players whose objectives are satisfied in $\mathsf{Outcome}(\pi)$. Formally, $i \in \mathsf{Win}(\pi)$ iff $\tau(\mathsf{Outcome}(\pi)) \in L(\psi_i)$. The set of *losers* in π, denoted $\mathsf{Lose}(\pi)$, is then $[k] \setminus \mathsf{Win}(\pi)$, namely the set of players whose objectives are not satisfied in $\mathsf{Outcome}(\pi)$.

A game $\mathcal{G}$ is *zero-sum* if the objectives of the players form a partition of all possible behaviors. That is, for every $i \neq j \in [k]$, we have that $L(\psi_i) \cap L(\psi_j) = \emptyset$, and $\bigcup_{i \in [k]} L(\psi_i) = (2^{AP})^\omega$. Accordingly, for every profile π in a zero-sum game, we have that $|\mathsf{Win}(\pi)| = 1$ and $|\mathsf{Lose}(\pi)| = k-1$. We then say that Player i *wins* $\mathcal{G}$ if she has a *winning strategy* – a strategy that guarantees the satisfaction of ψ_i no matter how the other players proceed. Formally, f_i is a winning strategy if for every profile π with f_i, we have that $\mathsf{Win}(\pi) = \{i\}$.

Games may be *non zero-sum*, thus the objectives of the players may overlap. In such games, we are interested in *stable* profiles. In particular, a profile $\pi = \langle f_1, \ldots, f_k \rangle$ is a *Nash Equilibrium* (NE, for short) [21] if, intuitively, no (single) player can benefit from unilaterally changing her strategy. In our setting, benefiting amounts to moving from the set of losers to the set of winners. Formally, for $i \in [k]$ and a strategy f'_i for Player i, let $\pi[i \leftarrow f'_i] = \langle f_1, \ldots, f_{i-1}, f'_i, f_{i+1}, \ldots, f_k \rangle$ be the profile obtained from π by changing the strategy of Player i to f'_i. We say that π is an NE if for every $i \in [k]$, if $i \in \mathsf{Lose}(\pi)$, then for every strategy f'_i, we have that $i \in \mathsf{Lose}(\pi[i \leftarrow f'_i])$. Thus, π is an NE if no player has an incentive to deviate from π. For a subset $\mathsf{W} \subseteq [k]$ of players, we say that π is a W-*NE* if π is an NE with $\mathsf{W} = \mathsf{Win}(\pi)$.

The game $\mathcal{G}$ is *turn-based* if the transition function of its graph G is such that for every vertex $v \in V$, there is a single player that "owns" v and determines the successor vertex whenever the play is in v. Formally, for every $v \in V$, there is $i \in [k]$ such that for all moves $\langle a_1, \ldots, a_k \rangle$ and $\langle a'_1, \ldots, a'_k \rangle$ that are possible for v, if $a_i = a'_i$, then $\delta(v, \langle a_1, \ldots, a_k \rangle) = \delta(v, \langle a'_1, \ldots, a'_k \rangle)$. Accordingly, we describe the game graph of a turn-based game as $G = \langle AP, \{V_i\}_{i \in [k]}, v_0, E, \tau \rangle$, where $V_1, \ldots, V_k$ is a partition of V to the sets of vertices owned by the different players, and $E \subseteq V \times V$ is the transition relation, modeling the fact that the set of actions of Player i in a vertex v she owned is the set of v's successors.

3 Rational Synthesis

Consider a k-player game $\mathcal{G} = \langle G, \{\psi_i\}_{i \in [k]} \rangle$. We distinguish between three types of players: A player is *controllable* if she is guaranteed to follow a strategy assigned to her. Otherwise, she is *uncontrollable*. The uncontrollable players are *rational* – they would not deviate from a profile unless they have a beneficial deviation from it. We distinguish between *cooperative uncontrollable* players, to which we can suggest a strategy (which they would follow unless they have a beneficial deviation), and *non-cooperative uncontrollable* players, to which we

cannot suggest a strategy. The distinction between the cooperative and non-cooperative uncontrollable players may be induced by the architecture or the nature of the players. We denote by C, CU, and NU the disjoint partition of $[k]$ into the classes of controllable, uncontrollable cooperative, and uncontrollable non-cooperative players, respectively.

In rational synthesis, we seek a strategy for each of the players in C with which their objectives are guaranteed to be satisfied, assuming rationality of the other players. As we have the best interest of the players in C in mind, we assume that $\mathsf{C} \neq \emptyset$. We say that a profile $\pi = \langle f_1, \ldots, f_k \rangle$ is a C-*fixed NE*, if no player in $\mathsf{CU} \cup \mathsf{NU}$ has a beneficial deviation. Formally, we have the following.

Definition 1. [Rational Synthesis] *Consider a k-player game $\mathcal{G} = \langle G, \{\psi_i\}_{i \in [k]} \rangle$. The problem of rational synthesis (RS) is to return a $(\mathsf{C} \cup \mathsf{CU})$-profile π' such that there is a C-fixed NE that extends π', and for every C-fixed NE π that extends π', we have that $\mathsf{C} \subseteq \mathsf{Win}(\pi)$.*

Two special cases of rational synthesis have been studied in the literature. The first is *cooperative rational synthesis*, where all uncontrollable players are cooperative [12]. The second is *non-cooperative rational synthesis*, where all uncontrollable players are non-cooperative [15].

Definition 2. [Cooperative Rational Synthesis] *Consider a k-player game $\mathcal{G} = \langle G, \{\psi_i\}_{i \in [k]} \rangle$ with $\mathsf{NU} = \emptyset$. The problem of cooperative rational synthesis (CRS) is to return a C-fixed NE π such that $\mathsf{C} \subseteq \mathsf{Win}(\pi)$.*

Definition 3. [Non-Cooperative Rational Synthesis] *Consider a k-player game $\mathcal{G} = \langle G, \{\psi_i\}_{i \in [k]} \rangle$ with $\mathsf{CU} = \emptyset$. The problem of non-cooperative rational synthesis (NRS) is to return a C-profile π' such that there is a C-fixed NE that extends π', and for every C-fixed NE π that extends π', we have that $\mathsf{C} \subseteq \mathsf{Win}(\pi)$.*

Remark 1. The original rational synthesis problem does not include a game graph [12]. Instead, the set AP over which the objectives are defined is partitioned among the players, and at each round of the game, each player chooses an assignment to the subset of AP she controls. It is easy to see that this setting is a special case of our setting, taking the graph to have vertices in 2^{AP}. □

Remark 2. In previous work, the definition of NRS does not require the existence of a C-fixed NE that extends π' [15, 18]. In some settings (in particular, turn-based games), the existence of such an NE is guaranteed. In others (in particular, concurrent games) there need not be an NE in games with three or more players [8]. Note, however, that even with the requirement that a C-fixed NE that extends π' exists, there is no guarantee that best response dynamics from π' would lead to such a C-fixed NE. □

As in traditional synthesis, one can also define the corresponding decision problems, of *rational realizability*, where we only need to decide whether the desired strategies exist. In order to avoid additional notations, we sometimes refer to RS, CRS, and NRS also as decision problems.

For a set $\mathsf{W} \subseteq [k]$, we say that a solution to the rational synthesis problem is a W-*solution* iff it is a solution that guarantees the winning of exactly the players in W. In particular, a $(\mathsf{C} \cup \mathsf{CU})$-profile π' is a W-*RS solution* if it is an RS solution such that for every C-fixed NE π that extends π', we have that $\mathsf{W} = \mathsf{Win}(\pi)$; a profile π is a W-*CRS solution* if π is a CRS solution such that $\mathsf{W} = \mathsf{Win}(\pi)$; and a C-profile π' is a W-*NRS solution* if π' is an NRS solution such that for every C-fixed NE π that extends π', we have that $\mathsf{W} = \mathsf{Win}(\pi)$.

It is easy to see that since the players in C are controllable, we can treat them as a single player with an objective that is the conjunction of the objectives of the players in C. Accordingly, in the sequel we assume that $\mathsf{C} = \{1\}$.

Remark 3. It is easy to add to the setting *uncontrollable hostile* players, namely players that, as in traditional synthesis, do not have an objective. Indeed, an uncontrollable hostile player is equivalent to an uncontrollable (cooperative or non-cooperative) player with objective $\neg\psi_1$. □

4 The Complexity of Cooperative Rational Synthesis

In this section we study the complexity of CRS. Consider a k-player concurrent game $\mathcal{G} = \langle G, \{\psi_i\}_{i\in[k]}\rangle$. A strategy for Player i can be viewed as an A_i-labeled V-tree, and a profile can be viewed as an $(A_1 \times \cdots \times A_k)$-labeled V-tree. Formally, if $\pi = \langle f_1, \ldots, f_k\rangle$ then for every node $h \in V^*$ in the *profile tree* $\langle V^*, \pi\rangle$, we have $\pi(h) = \langle f_1(h), \ldots, f_k(h)\rangle$, where $\langle V^*, f_i\rangle$ is the *strategy tree* that corresponds to f_i. Note that $\mathsf{Outcome}(\pi)$ then corresponds to a path in $\langle V^*, \pi\rangle$.

Viewing profiles as labeled trees enables us to reduce CRS to the nonemptiness of a tree automaton. Essentially, the automaton accepts all profile trees that are solutions to the CRS problem. We define the automaton by decomposing the solutions according to the set of players that win. Given a set W of players with $1 \in \mathsf{W}$, a profile π is a W-CRS solution iff π is a 1-fixed W-NE. Thus, iff exactly the players in W win in $\mathsf{Outcome}(\pi)$, and for every $i \notin \mathsf{W}$, Player i loses in $\mathsf{Outcome}(\pi[i \leftarrow f_i'])$, for every strategy f_i'. In Theorem 1 below, we construct automata that check these conditions.

Theorem 1. *Consider a set of players W with $1 \in \mathsf{W}$. We can construct the following tree automata over $(A_1 \times \cdots \times A_k)$-labeled V-trees:*

- *An NBT $\mathcal{N}_\mathsf{W}$ that accepts a profile tree $\langle V^*, \pi\rangle$ iff $\mathsf{Win}(\pi) = \mathsf{W}$. The size of $\mathcal{N}_\mathsf{W}$ is polynomial in $|G|$ and exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$.*
- *For every $i \notin \mathsf{W}$, a UCT $\mathcal{U}^i_\mathsf{W}$ that accepts a profile tree $\langle V^*, \pi\rangle$ iff $i \in \mathsf{Lose}(\pi[i \leftarrow f_i'])$, for every strategy f_i'. The size of $\mathcal{U}^i_\mathsf{W}$ is polynomial in $|G|$ and exponential in $|\psi_i|$.*

Theorem 2. *Solving CRS can be done in time polynomial in $|G|$, exponential in $|\psi_1|$, and doubly-exponential in $|\psi_2|, \ldots, |\psi_k|$. The problem is EXPTIME-hard in $|\psi_1|$ and 2EXPTIME-hard in each of $|\psi_2|, \ldots, |\psi_k|$.*

Proof. We start with the upper bound. It is easy to see that for every set $\mathsf{W} \subseteq [k]$ of players with $1 \in \mathsf{W}$, there is a W-CRS solution iff the intersection of the automata constructed in Theorem 1 is nonempty. We construct an NBT $\mathcal{A}$ such that $L(\mathcal{A}) \neq \emptyset$ iff $L(\mathcal{N}_\mathsf{W}) \cap \bigcap_{i \notin \mathsf{W}} L(\mathcal{U}^i_\mathsf{W}) \neq \emptyset$, and $|\mathcal{A}|$ is polynomial in $|G|$, exponential in $|\psi_1|$, and doubly-exponential in $|\psi_2|, \ldots, |\psi_k|$. Since nonemptiness of NBTs can be checked in quadratic time [26], the upper bound follows. Moreover, when $L(\mathcal{A}) \neq \emptyset$, the algorithm returns a witness to $\mathcal{A}$'s nonemptiness, namely a profile tree that is a solution to the CRS problem.

The construction of $\mathcal{A}$ involves two challenges. First, a naive analysis of the blow-up involved in translating UCTs to NBTs is exponential in the state space of the UCT. In our case, the state space of a UCT $\mathcal{U}^i_\mathsf{W}$ is of the form $V \times S$, for some set S that is independent of G. Also, the V-component is updated deterministically: all states sent to the same direction v of the tree agree on their V-element. Consequently, the exponential blow up is only in the S-component, which depends only on $|\psi_i|$. Second, the transformation of UCTs to NBTs that is described in [20] preserves nonemptiness, whereas here we need to preserve nonemptiness of an intersection of automata. As detailed in [17], where we coped with a similar challenge, this can be handled by parameterizing the construction in [20] by a rank (essentially, a bound on the size of transducers that generate trees in the language of the automaton) that corresponds to the size of the intersection.

We continue to the lower bounds, and we show they are valid already in the case $k = 2$. Proving an EXPTIME lower bound in $|\psi_1|$, we describe a reduction from the membership problem for linear-space alternating Turing machines (ATM), defined in Appendix B.2. That is, given an ATM M with space complexity $s : \mathbb{N} \to \mathbb{N}$ and a word w, we construct a 2-player turn-based game $\mathcal{G} = \langle G, \{\psi_1, \psi_2\}\rangle$, such that G and ψ_2 are of a fixed size, ψ_1 is of size linear in $s(|w|)$, and there is a CRS solution in $\mathcal{G}$ iff M accepts w.

Essentially, Player 1 and Player 2 generate a branch in the computation tree of M on w. Player 1 chooses the letters of the current configuration one by one, and chooses, at the end of each existential configuration, the successor configuration to which the branch continues. Player 2, on the other hand, only chooses successor configurations at the end of each universal configuration. The objective of Player 1 is to reach an accepting configuration, and the objective of Player 2 is to reach a rejecting configuration.

We prove that $\mathcal{G}$ has a $\{1\}$-NE that satisfies ψ_1 iff M accepts w. First, if M accepts w, then the profile in which Player 1 follows a strategy that generates the configurations in the accepting computation and chooses the appropriate successors to existential configurations, is a $\{1\}$-NE that satisfies ψ_1. Also, if M rejects w, then Player 2 can choose successors of universal configurations in a way that leads to a rejecting configuration. Thus, there is no $\{1\}$-NE in $\mathcal{G}$ that satisfies ψ_1, as either Player 1 loses by not forming a valid branch, or Player 2 can deviate to a strategy where she wins and Player 1 loses. In Appendix B.2, we give the details of the reduction.

Proving a 2EXPTIME lower bound in $|\psi_2|$, we use a reduction from decidability of 2-player zero-sum games, which is 2EXPTIME-hard already for a game with a game graph of a fixed size [2]. Given a 2-player zero-sum game $\mathcal{G} = \langle G, \psi \rangle$, we construct a 2-player game $\mathcal{H} = \langle H, \{\psi_1, \psi_2\} \rangle$ such that the size of H is linear in $|G|$, ψ_1 is of a fixed size, ψ_2 is of size linear in $|\psi|$, and there is a CRS solution in $\mathcal{H}$ iff Player 1 wins $\mathcal{G}$. Essentially, the game graph H contains two copies of G, and a new initial vertex in which Player 2 chooses between proceeding to the first or the second copy. Note that Player 1 has no influence in that decision. Then, the objective of Player 1 is for the play to be generated in the first copy, and the objective of Player 2 is for the play to be generated in the second copy and for the computation to not satisfy ψ. In Appendix B.3, we describe H, ψ_1, and ψ_2 formally. □

Remark 4. Note that our algorithm finds W-CRS solutions for all $\mathsf{W} \subseteq [k]$, and so it is exponential in k. As shown in [10], rational synthesis is PSPACE in k already for rational synthesis with reachability objectives.

5 The Complexity of Non-Cooperative Rational Synthesis

In this section we study the complexity of NRS. We start with the turn-based setting, and then proceed the concurrent setting.

5.1 Turn-based games

As in the CRS case, we construct a tree automaton that accepts strategy trees that are NRS solutions. Here, however, the trees are labeled not only by a strategy for Player 1, but also by information that certifies that the suggested strategy is indeed a solution. Our construction follows the ideas developed for turn-based games in [10], adding to them a treatment of the LTL objectives (the latter is not too complicated, and our main goal in this section is to set the stage to the concurrent setting, which was left open in [10]). In order to present our solution, we first need some definitions and notations.

Consider a k-player turn-based game $\mathcal{G} = \langle G, \{\psi_i\}_{i \in [k]} \rangle$. Let $G = \langle AP, \{V_i\}_{i \in [k]}, v_0, E, \tau \rangle$. Recall that for a subset $S \subseteq [k]$ of players, an *S-profile* is a set of strategies, one for each player in S, and that a profile π *extends* an S-profile π' if the players in S use in π their strategies in π'. The outcome of an S-profile π', denoted $\mathsf{Outcome}(\pi')$, is the union of plays that are outcomes of profiles that extend π'. Thus, $\mathsf{Outcome}(\pi') \subseteq V^\omega$ is the set of plays that are possible outcome of the game when the players in S follow their strategies in π'.

Consider a profile $\pi = \langle f_1, \ldots, f_k \rangle$ and a prefix $h \in V^*$ of $\mathsf{Outcome}(\pi)$. For a profile $\pi' = \langle f'_1, \ldots, f'_k \rangle$, we define the profile $\mathsf{switch}(\pi, \pi', h) = \langle f_1^h, \ldots, f_k^h \rangle$ as the profile in which the players first follow π and generate h, and then switch to following π'. Formally, for every $x \in V^*$ and Player $i \in [k]$, if $x \preceq h$, then $f_i^h(x) = f_i(x)$, and if $x = h \cdot y$, then $f_i^h(x) = f'_i(y)$. Note that since the last vertices in x and y coincide, then $\mathsf{switch}(\pi, \pi', h)$ is well defined, in the sense that

it returns only allowed actions. Also note that $f_i^h(h) = f_i'(\varepsilon)$, thus, switching to following π', we reset the history of the game so far. The strategies in nodes that are neither a prefix of h nor an extension of h are arbitrary and can follow π.

For $i \in [k] \setminus \{1\}$ and a prefix $h \in V^* \cdot V_i$ of some play in $\mathsf{Outcome}(\{f_1\})$, we say that *Player i wins from h* if there exists a strategy f_i' for Player i such that for every profile $\pi = \langle f_1, \ldots, f_k \rangle$ with $h \preceq \mathsf{Outcome}(\pi)$, we have that $i \in \mathsf{Win}(\mathsf{switch}(\pi, \pi[i \leftarrow f_i'], h))$. That is, Player i wins in every profile in which the players first generate h, and then Player i switches to following f_i', while the other players adhere to their strategies in the original profile. The strategy f_i' is then called an *h-winning strategy for Player i*.

Since turn-based games always have an NE, an NRS solution in $\mathcal{G}$ is a strategy f_1 for Player 1 such that for every 1-fixed NE $\pi = \langle f_1, \ldots, f_k \rangle$, we have that $1 \in \mathsf{Win}(\pi)$. Equivalently, for every profile $\pi = \langle f_1, \ldots, f_k \rangle$, we have that either $1 \in \mathsf{Win}(\pi)$, or there exists $i \in \mathsf{Lose}(\pi)$ such that $i \in \mathsf{Win}(\pi[i \leftarrow f_i'])$ for some strategy f_i' for Player i. As detailed in [10], this implies that a strategy f_1 for Player 1 is an NRS solution iff for every path ρ in $\mathsf{Outcome}(\{f_1\})$, either $\tau(\rho) \in L(\psi_1)$, or there is $i \in [k] \setminus \{1\}$ such that $\tau(\rho) \notin L(\psi_i)$, and there are a prefix $h \preceq \rho$ and an h-winning strategy f_i' for Player i. We then say that h is a *good deviation point* for Player i, and f_i' is a *good deviation* for Player i.

Our goal is to define a tree automaton that accepts a strategy tree for Player 1 iff it is an NRS solution. The tree automaton should check that every path in $\mathsf{Outcome}(\{f_1\})$ that does not satisfy ψ_1 has a good deviation point for one of the players that lose in it. For that purpose, a strategy f_1 of Player 1 is going to be *certified* by information about deviations: each path in $\mathsf{Outcome}(\{f_1\})$ that does not satisfy ψ_1 is labeled by a player i that loses in the path, a good deviation point for Player i, and a good deviation for Player i. Note that each deviation may handle only a subset of the paths below the good deviation point, and thus a subtree in the certified strategy tree may be labeled by strategies of different players, each deviating at different points.

Formally, a certified strategy tree is a $((V \cup \{\oslash\}) \times [k])$-labeled V-tree $\langle V^*, g \rangle$, where each node is labeled by a pair $\langle v, i \rangle$, where $v \in V \cup \{\oslash\}$ is a *strategy-label*, and $i \in [k]$ is a *player-label*. We use g_s and g_p to refer to the projection of g on the strategy and player components. Each path in the tree that corresponds to a play in $\mathsf{Outcome}(\{f_1\})$ has a suffix all whose nodes are labeled by the same player label. If this label is 1, then the strategy labels describe a strategy of Player 1 and the path should satisfy ψ_1. If this label is $i \in [k] \setminus \{i\}$, then a deviation point of Player i has been encountered, the strategy labels describe a good deviation for Player i, and the path should not satisfy ψ_i. As long as a deviation point has not been encountered, the strategy labels describe a strategy for Player 1 (and so, they are in V in nodes with a direction in V_1, and are $\oslash$ in nodes with a direction not in V_1). Once a deviation point for Player i is encountered (which is indicated by the strategy label being changed from $\oslash$ to a vertex in V in a node with direction V_i), the strategy labels describe a strategy for Player i.

By adjusting Lemma 7 in [10] to the setting with LTL objectives, we get the following.

Theorem 3. *A strategy f_1 for Player 1 is an NRS solution iff there is a certified strategy tree $\langle V^*, g\rangle$ that agrees with f_1. Thus, for every $h \in V^*$ and $v \in V_1$ such that $h \cdot v \in \mathsf{Outcome}(\{f_1\})$, we have that $g_s(h \cdot v) = f_1(h \cdot v)$*

We now define a tree automaton that accepts certified strategy trees, which we then use for solving NRS.

Theorem 4. *We can construct a UCT $\mathcal{U}$ over $((V \cup \{\oslash\}) \times [k])$-labeled V-trees such that $\mathcal{U}$ accepts a $((V \cup \{\oslash\}) \times [k])$-labeled V-tree $\langle V^*, g\rangle$ iff $\langle V^*, g\rangle$ is a certified strategy tree. The size of $\mathcal{U}$ is polynomial in $|G|$ and exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$.*

Proof. The requirements on a certified strategy tree $\langle V^*, g\rangle$ for Player 1 can be decomposed to the following conditions.

- ($\mathbf{C_1^i}$) For every $i \in [k] \setminus \{1\}$, the subtree of every node $h \in V^* \cdot V_i$ in the tree that is labeled by $V \times [k]$ is labeled by an h-winning strategy for Player i.
- ($\mathbf{C_2}$) The (infinite) suffix of every path in the tree is p-labeled by a single $i \in [k]$.
- ($\mathbf{C_3^i}$) For every $i \in [k] \setminus \{1\}$, every path in the tree with a suffix p-labeled by i has a good deviation point for Player i.
- ($\mathbf{C_4^1}$) Player 1 wins in every path in the tree with suffix p-labeled by 1.
- ($\mathbf{C_4^i}$) for every $i \in [k] \setminus \{1\}$, Player i loses in every path in the tree with suffix p-labeled by i.

In Appendix B.4, we describe UCTs that check these conditions and whose intersection is of the desired size. □

Theorem 5. *Solving NRS can be done in time polynomial in $|G|$ and doubly-exponential in $|\psi_1|, \ldots, |\psi_k|$. The problem is 2EXPTIME-hard in each of $|\psi_1|, \ldots, |\psi_k|$.*

Proof. We start with the upper bound. By Theorems 3 and 4, we can reduce NRS to nonemptiness of a UCT $\mathcal{U}$ over $((V \cup \{\oslash\}) \times [k])$-labeled V-trees of size polynomial in $|G|$ and exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$. Using considerations similar to these used in the proof of Theorem 2 (in particular, the fact $\mathcal{U}$ is deterministic in its V-element), we can construct from it an NBT $\mathcal{N}$ of size polynomial in $|G|$ and doubly-exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$ that preserves the nonemptiness of $\mathcal{U}$. Since the nonemptiness problem for NBT can be solved in quadratic time [26], the desired complexity follows.

We continue to the lower bounds, and we show they are valid already in the case $k = 2$. We again use reductions from deciding 2-player zero-sum games. In order to prove 2EXPTIME-hardness in $|\psi_1|$, consider a 2-player zero-sum game $\mathcal{G} = \langle G, \psi\rangle$, for a fixed-size G. We claim that the 2-player game $\mathcal{G}' = \langle G, \{\psi, \mathbf{true}\}\rangle$ is such that G is of a fixed size and that there is an NRS solution in $\mathcal{G}'$ iff Player 1 wins $\mathcal{G}$. Indeed, since the objective of Player 2 is **true**, every profile π in $\mathcal{G}'$ is a 1-fixed NE. So, in order for a strategy f_1 to be an NRS solution, it must satisfy that $1 \in \mathsf{Win}(\langle f_1, f_2\rangle)$, for every strategy f_2 for Player 2. Equivalently, it is a winning strategy for Player 1 in $\mathcal{G}$.

In order to prove 2EXPTIME-hardness in $|\psi_2|$, consider again a 2-player zero-sum game $\mathcal{G} = \langle G, \psi \rangle$, for a fixed-size G. We construct a 2-player game $\mathcal{G}' = \langle H, \{\psi_1, \psi_2\} \rangle$ such that H and ψ_1 are of a fixed size, the size of ψ_2 is linear in $|\psi|$, and there is an NRS solution in $\mathcal{G}'$ iff Player 1 wins $\mathcal{G}$. The game graph H is as in the proof of Theorem 2. Thus, it has an initial vertex from which Player 2 chooses between two copies of G. The states of the first copy are labeled by a fresh atomic proposition p. Then, $\psi_1 = XGp$, and $\psi_2 = X((\psi \wedge Gp) \vee ((\neg\psi) \wedge G\neg p))$. Thus, the objective of Player 1 is for the play to be generated in the first copy, and the objective of Player 2 is either to generate a play in the first copy whose computation satisfies ψ, or to generate a play in the second copy whose computation does not satisfy ψ.

If Player 1 has a winning strategy f_1 in $\mathcal{G}$, then there is an NRS solution f_1' in $\mathcal{G}'$, where f_1' follows f_1 in the copy Player 2 chooses. Indeed, as f_1' guarantees the satisfaction of ψ for all possible behaviors of Player 2, a profile π is a 1-fixed NE only if Player 2 chooses the first copy. If Player 1 loses $\mathcal{G}$, then for every strategy f_1 for Player 1, there is a strategy f_2 for Player 2 such that $\mathsf{Outcome}(\langle f_1, f_2 \rangle)$ does not satisfy ψ. So, for every strategy f_1 for Player 1 in $\mathcal{G}'$, we have that there is a 1-fixed NE $\pi = \langle f_1, f_2 \rangle$ such that $1 \in \mathsf{Lose}(\pi)$, where f_2 is the strategy that chooses the second copy, and ensures that ψ is not satisfied. Hence, there is no NRS solution in $\mathcal{G}'$. □

5.2 Concurrent games

Consider a k-player concurrent game $\mathcal{G} = \langle G, \{\psi_i\}_{i \in [k]} \rangle$. Let $G = \langle AP, V, v_0, \{A_i\}_{i \in [k]}, \{\kappa_i\}_{i \in [k]}, \delta, \tau \rangle$. As our constructions in this section are loaded with notations, we simplify the setting and assume that there is one set A of actions, available to all players in all vertices. That is, $A_1 = A_2 = \cdots = A_k = A$, and for every $v \in V$ and $i \in [k]$, we have that $\kappa_i(v) = A$. All our constructions and results can be easily extended to the general case.

As in the turn-based setting, we define a UCT that accepts certified strategy trees for Player 1. In the concurrent setting, however, certification is much more complicated. Below we explain the challenges in the concurrent setting and how we overcome them. For $i \in [k] \setminus \{1\}$ and a prefix $h \in V^*$ of some path in $\mathsf{Outcome}(\{f_1\})$, we say that *Player i wins from h* if for every profile $\pi = \langle f_1, \ldots, f_k \rangle$ with $h \preceq \mathsf{Outcome}(\pi)$, we have that there exists a strategy f_i' for Player i such that $i \in \mathsf{Win}(\mathsf{switch}(\pi, \pi[i \leftarrow f_i'], h))$. Thus, Player i wins from h if she has a beneficial deviation to switch to from h, for every profile π with $h \preceq \mathsf{Outcome}(\pi)$. Note that for different profiles, Player i might have different such beneficial deviations. Here, however, the prefix h need not end in V_i (in fact, there is no V_i in the concurrent setting). We say that h is a *winning point* for Player i. Also, we say that $(h, \langle f_1(h), \ldots, f_k(h) \rangle)$ is a *good deviation pair* for Player i iff there exists $a_i' \in A$ such that $h \cdot \delta(h, \langle f_1(h), \ldots, a_i', \ldots, f_k(h) \rangle)$ is a winning point for Player i.

In order to understand better the difference between NRS solutions in the turn-based and concurrent settings, recall that a strategy f_1 for Player 1 is not an NRS solution iff there is a 1-fixed NE $\pi = \langle f_1, \ldots, f_k \rangle$ whose outcome ρ does not

satisfy ψ_1. Note that π being a 1-fixed NE means that for every prefix $h \cdot v \cdot u$ of ρ, there exist actions $\langle a_2, \ldots, a_k \rangle \in A^{k-1}$ such that $\delta(v, \langle f_1(h \cdot v), a_2, \ldots, a_k \rangle) = u$ and for every $i \in \mathsf{Lose}(\rho)$, we have that $(h \cdot v, \langle f_1(h \cdot v), a_2, \ldots, a_k \rangle)$ is not a good deviation pair for Player i. In particular, we can choose $a_i = f_i(h \cdot v)$. Hence, if f_1 is an NRS solution, and there exists a path $\rho \in \mathsf{Outcome}(\{f_1\})$ that does not satisfy ψ_1, then there must be a prefix $h \cdot v \cdot u \preceq \rho$ such that for every $\langle a_2, \ldots, a_k \rangle \in A^{k-1}$ with $\delta(v, \langle f_1(h \cdot v), a_2, \ldots, a_k \rangle) = u$, there exists $i \in \mathsf{Lose}(\rho)$ such that $(h \cdot v, \langle f_1(h \cdot v), a_2, \ldots, a_k \rangle)$ is a good deviation point for Player i. We then say that $h \cdot v \cdot u$ is a *good deviation transition for* $\mathsf{Lose}(\rho)$. Thus, while in the turn-based settings it is sufficient to find in every path in which Player 1 loses a good deviation point for one of the players that lose in it, in the concurrent setting the definition of good deviation depends on the transition induced by the specific profile being used, and so we have to consider deviating transitions, and there may be several players in $\mathsf{Lose}(\rho)$ that deviate. Accordingly, in order to certify a strategy for Player 1, we should describe a mapping from every vector of actions to a set of players, along with their deviations.

Another difference between the turn-based setting and the concurrent setting is that only in the first, the existence of some 1-fixed NE is guaranteed [8]. Hence, we have to add to the algorithm such a check (which is in fact easy).

We can now define certified strategy trees for the concurrent setting. Every node in a certified strategy tree is labeled by the following components:

1. An action $a_1 \in A$, which is the strategy for Player 1.
2. A *deviation function* $d : A^{k-1} \to (A \cup \{\bot\})^{k-1}$, which maps a vector of actions of players $2, \ldots, k$ to the set of players that deviate from it, along with their deviations. Specifically, $\langle a_2, \ldots, a_k \rangle \in A^{k-1}$ being mapped to $\langle a'_2, \ldots, a'_k \rangle \in (A \cup \{\bot\})^{k-1}$ indicates that for every $i \in [k] \setminus \{1\}$, if $a'_i \in A$, then a'_i is the deviation for Player i from $\langle a_2, \ldots, a_k \rangle$, and if $a'_i = \bot$ then no deviation from $\langle a_2, \ldots, a_k \rangle$ is specified for Player i. Let $\mathcal{D}$ denote the set of all possible deviation functions.
3. A set $\mathsf{L} \subseteq \{2, \ldots, k\}$ of players, which describes the set of players that lose in a given path and which are therefore expected to have a good deviation transition. That is, if a suffix of a path is labeled by $\emptyset$, then Player 1 should win in this path, and if a suffix of a path is labeled by $\mathsf{L} \neq \emptyset$, then all the players in L lose in this path.
4. A vector of actions $\langle a_2, \ldots, a_k \rangle \in A^{k-1}$, which describes the strategies for the other players in the required 1-fixed NE.

Formally, a certified strategy tree is a $(A \times \mathcal{D} \times 2^{\{2,\ldots,k\}} \times A^{k-1})$-labeled V-tree $\langle V^*, g \rangle$, where each node is labeled by both a *strategy-label* $a_1 \in A$, a *deviation-label* $d \in \mathcal{D}$, a *player-label* $\mathsf{L} \in 2^{\{2,\ldots,k\}}$, and an *NE-label* $\langle a_2, \ldots, a_k \rangle \in A^{k-1}$. We use g_s, g_d, g_p, and g_{NE} to refer to the projection of g on its different components.

For a node $h \cdot v$ that is s-labeled by a_1 and d-labeled by d, a possible successor u of v, and a set of losers L, we say that $h \cdot v \cdot u$ is *marked as a good deviation transition for* L iff the following hold:

1. For every $\langle a_2, \ldots, a_k \rangle \in A^{k-1}$ such that $\delta(v, \langle a_1, a_2, \ldots, a_k \rangle) = u$, there exists $i \in \mathsf{L}$ such that $(d(\langle a_2, \ldots, a_k \rangle))_i \in A$. That is, for every vector of

actions $\langle a_2, \ldots, a_k \rangle$ that leads to u, there is $i \in \mathsf{L}$ such that d assigns a deviation for Player i from $\langle a_2, \ldots, a_k \rangle$.
2. For every $i \in \mathsf{L}$, there is a vector of actions $\langle a_2, \ldots, a_k \rangle \in A^{k-1}$ such that $\delta(v, \langle a_1, a_2, \ldots, a_k \rangle) = u$, and $(d(\langle a_2, \ldots, a_k \rangle))_i \in A$. That is, for every $i \in \mathsf{L}$ there exists a vector of actions that leads to u, from which d assigns a deviation for Player i.

Now, an $(A \times \mathcal{D} \times 2^{\{2,\ldots,k\}} \times A^{k-1})$-labeled V-tree $\langle V^*, g \rangle$ is a certified strategy tree iff it satisfies the following conditions:

- $(\mathbf{C_1})$ If $h \cdot v \cdot u \in V^*$ is marked as a good deviation transition for a set of players $\mathsf{L} \subseteq \{2, \ldots, k\}$, $\mathsf{L} \neq \emptyset$, then it is indeed a good deviation transition for L.
- $(\mathbf{C_2})$ Every path ρ has a set L such that ρ is eventually always p-labeled by L.
- $(\mathbf{C_3^L})$ For every $\mathsf{L} \subseteq \{2, \ldots, k\}$, $\mathsf{L} \neq \emptyset$, every path in the tree with a suffix p-labeled by L has a good deviation transition for L.
- $(\mathbf{C_4^1})$ Player 1 wins in every path in the tree with suffix p-labeled by $\emptyset$.
- $(\mathbf{C_4^i})$ For every $i \in [k] \setminus \{1\}$, Player i loses in every path in the tree with suffix p-labeled by L such that $i \in \mathsf{L}$.
- $(\mathbf{C_5})$ The s and NE-labeling of the tree specifies a 1-fixed NE.

Theorem 6. *A strategy f_1 for Player 1 is an NRS solution iff there is a certified strategy tree $\langle V^*, g \rangle$ that agrees with f_1. That is, for every $h \in V^*$, we have that $g_s(h) = f_1(h)$.*

Theorem 7. *We can construct a UCT over $(A \times \mathcal{D} \times 2^{\{2,\ldots,k\}} \times A^{k-1})$-labeled V-trees that accepts a $(A \times \mathcal{D} \times 2^{\{2,\ldots,k\}} \times A^{k-1})$-labeled V-tree $\langle V^*, g \rangle$ iff $\langle V^*, g \rangle$ is a certified strategy tree. The size of the UCT is polynomial in $|G|$ and exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$.*

Proof. We can construct UCTs for $\mathbf{C_2}$, $\mathbf{C_3^L}$, and $\mathbf{C_4^i}$ that are very similar to the UCTs for $\mathbf{C_2}$, $\mathbf{C_3^i}$ and $\mathbf{C_4^i}$ from the turn-based setting. The UCT for $\mathbf{C_5}$ is similar to the UCT for CRS solutions. In Appendix B.5, we describe in detail a UCT that checks the satisfaction of $\mathbf{C_1}$. Then, the conjunction of the above UCTs results in a UCT that accepts certified strategy trees. □

The number of deviation functions $d : A^{k-1} \to (A \cup \{\bot\})^{k-1}$ is exponential in $|A|$. Consequently, the UCT described in Theorem 7 has an exponential alphabet, which causes the NBT generated in [20] to have exponentially many transitions, making its nonemptiness problem exponential. In order to overcome this problem, we introduce *vertical annotation of certified strategy trees*, which essentially replace a node labeled by $d \in \mathcal{D}$ by a sequence of nodes, labeled by a smaller alphabet.

Explaining our vertical annotation, we find it clearer to go back to a detailed description of the actions of the different players, thus refer to A_i and κ_i rather than assuming they are all equal to A. For every vertex $v \in V$ and an action

$a_1 \in \kappa_1(v)$ for Player 1, we denote by T_{v,a_1} the set of possible vectors of actions from v, given Player 1 chose a_1. That is, $T_{v,a_1} = \{a_1\} \times \kappa_2(v) \times \cdots \times \kappa_k(v)$. We order the vectors in T_{v,a_1} arbitrarily, and, for $1 \leq i \leq |T_{v,a_1}|$, denote by t^i_{v,a_1} the i-th item in T_{v,a_1}. We also denote by $t^i_{v,a_1}[j \leftarrow a'_j]$ the vector of actions obtained from t^i_{v,a_1} by replacing the action for Player j by a'_j.

For a given transition from v to v', let T_{v,v',a_1} be the restriction of T_{v,a_1} to vectors $\langle a_1, a_2, \ldots, a_k\rangle$ such that $\delta(v, \langle a_1, a_2, \ldots, a_k\rangle) = v'$, and let t^i_{v,v',a_1} denote the i-th item in T_{v,v',a_1}. Also, let $T = \bigcup_{v\in V} \bigcup_{a_1 \in \kappa_1(v)} T_{v,a_1}$, and $\Sigma = A_1 \cup ((A_2 \times \cdots \times A_k) \cup \bigcup_{j\in \mathsf{L}}(\{j\} \times A_j)) \times (\emptyset \cup (V \times 2^{\{2,\ldots,k\}}))$. We also need an additional $2^{\{2,\ldots,k\}}$ component, for annotating the set of losers in each path in the tree, but we omit it for now, as it is not relevant for the vertical annotations.

A certified strategy tree is now a Σ-labeled $(V \cup T)$-tree, where nodes with direction in V are labeled by the strategy for Player 1, and nodes with direction in T are labeled with deviation information. Consider a node that corresponds to a vertex v and is labeled by a_1. Following vertically there are nodes corresponding to T_{v,a_1}, each labeled by a vector of actions. This information is for verifying good deviation transitions. That is, after a good deviation transition is announced, we need to verify that the involved players indeed have appropriate beneficial deviations. The last node in the chain, which corresponds to $t^{|T_{v,a_1}|}_{v,a_1}$, is either not labeled, or labeled by a vertex v' and a set L of losers. If it is not labeled, it means there are no good deviation transitions from v, in which case we continue to the nodes corresponding to the possible successors of v. If it is labeled by $\langle v', \mathsf{L}\rangle$, it means that v, v' is a good deviation transition for L. Hence, the following nodes correspond to T_{v,v',a_1}, each labeled by a single deviation, followed by a chain of good deviation transitions, using the appropriate $V \times 2^{\{2,\ldots,k\}}$ annotations, or by nodes corresponding to successor vertices. In Appendix B.6, we describe formally a UCT for verifying good deviation transitions in vertically annotated certified strategy trees.

Theorem 8. *Solving NRS in the concurrent setting can be done in time polynomial in $|G|$ and doubly-exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$. The problem is 2EXPTIME-hard in each of $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$.*

Proof. We start with the upper bound. We can easily modify the UCTs for $\mathbf{C_2}$, $\mathbf{C^L_3}$, $\mathbf{C^i_4}$, and $\mathbf{C_5}$ from the proof of Theorem 7 to accommodate the vertical annotation. With conjunction with the UCT for verifying good deviation transitions, described in Appendix B.6, we have a UCT $\mathcal{U}$ over Σ-labeled $(V \cup T)$-trees such that $\mathcal{U}$ accepts a Σ-labeled $(V \cup T)$-tree $\langle V^*, g\rangle$ iff $\langle V^*, g\rangle$ is a vertically annotated certified strategy tree. By Theorem 6, there is an NRS solution in $\mathcal{G}$ iff $\mathcal{U}$ is not empty. The size of $\mathcal{U}$ is polynomial in $|G|$ and exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$, and its alphabet is polynomial in G. Also, $\mathcal{U}$ is deterministic in its V-element. Hence, as detailed in the proof of Theorem 2, we can construct from $\mathcal{U}$ an NBT $\mathcal{N}$ of size polynomial in $|G|$ and doubly-exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$ such that $L(\mathcal{U}) \neq \emptyset$ iff $L(\mathcal{N}) \neq \emptyset$. Since the nonemptiness problem for NBT can be solved in quadratic time [26], the desired complexity follows.

Finally, as turn-based games are a special case of concurrent ones, the lower bound from Theorem 5 applies here. □

5.3 General rational synthesis

Consider a k-player game $\mathcal{G} = \langle G, \{\psi_i\}_{i\in[k]}\rangle$. Recall that the problem of rational synthesis is to return a $(\{1\} \cup \mathsf{CU})$-profile π' such that there is a 1-fixed NE that extends π', and for every 1-fixed NE π that extends π', we have that $1 \in \mathsf{Win}(\pi)$.

A $(\{1\}\cup\mathsf{CU})$-profile π' is an RS-solution iff for every path ρ such that $\rho \not\models \psi_1$, there is no 1-fixed NE π that extends π' and $\mathsf{Outcome}(\pi) = \rho$. It is easy to see that we can define certified $(\{1\} \cup \mathsf{CU})$-profile trees in a similar way we defined certified strategy trees for Player 1, inducing an algorithm of the same complexity for checking the existence of a certified $(\{1\} \cup \mathsf{CU})$-profile tree. Also, as NRS is a special case of RS, the NRS lower bound provide a lower bound for RS. Hence, we can conclude with the following.

Theorem 9. *Solving RS can be done in time polynomial in $|G|$ and doubly-exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$. The problem is 2EXPTIME-hard in each of $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$.*

References

1. S. Almagor, O. Kupferman, and G. Perelli. Synthesis of controllable Nash equilibria in quantitative objective game. In *Proc. 27th Int. Joint Conf. on Artificial Intelligence*, pages 35–41, 2018.
2. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.
3. E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Proc. 45th IEEE Symp. on Foundations of Computer Science*, pages 295–304. IEEE Computer Society, 2004.
4. P. Bouyer-Decitre, O. Kupferman, N. Markey, B. Maubert, A. Murano, and G. Perelli. Reasoning about quality and fuzziness of strategic behaviours. In *Proc. 28th Int. Joint Conf. on Artificial Intelligence*, pages 1588–1594, 2019.
5. A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.
6. K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proc. 19th Int. Conf. on Concurrency Theory*, volume 5201 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2008.
7. K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *Proc. 18th Int. Conf. on Concurrency Theory*, pages 59–73, 2007.
8. K. Chatterjee, R. Majumdar, and M. Jurdzinski. On Nash equilibria in stochastic games. In *Proc. 13th Annual Conf. of the European Association for Computer Science Logic*, volume 3210 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2004.
9. A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.

10. R. Condurache, E. Filiot, R. Gentilini, and J.-F. Raskin. The complexity of rational synthesis. In *Proc. 43th Int. Colloq. on Automata, Languages, and Programming*, volume 55 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
11. E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 328–337, 1988.
12. D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *Proc. 16th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010.
13. J. Gutierrez, G. Perelli, and M. J. Wooldridge. Imperfect information in reactive modules games. *Inf. Comput.*, 261:650–675, 2018.
14. O. Kupferman, Y. Lustig, M.Y. Vardi, and M. Yannakakis. Temporal synthesis for bounded systems and environments. In *Proc. 28th Symp. on Theoretical Aspects of Computer Science*, pages 615–626, 2011.
15. O. Kupferman, G. Perelli, and M.Y. Vardi. Synthesis with rational environments. *Annals of Mathematics and Artificial Intelligence*, 78(1):3–20, 2016.
16. O. Kupferman and N. Piterman. Lower bounds on witnesses for nonemptiness of universal co-Büchi automata. In *Proc. 12th Int. Conf. on Foundations of Software Science and Computation Structures*, volume 5504 of *Lecture Notes in Computer Science*, pages 182–196. Springer, 2009.
17. O. Kupferman and N. Shenwald. Perspective games with notifications. In *Proc. 40th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 182 of *LIPIcs*, pages 51:1–51:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
18. O. Kupferman and N. Shenwald. Perspective multi-player games. Submitted, 2021.
19. O. Kupferman and G. Vardi. Perspective games. In *Proc. 34th IEEE Symp. on Logic in Computer Science*, pages 1 – 13, 2019.
20. O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.
21. J.F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*, 1950.
22. N. Nisan, T. Roughgarden, E. Tardos, and V.V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.
23. C. H. Papadimitriou. Algorithms, games, and the internet. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 749–753, 2001.
24. A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
25. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.
26. M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and Systems Science*, 32(2):182–221, 1986.
27. M. Wooldridge, J. Gutierrez, P. Harrenstein, E. Marchioni, G. Perelli, and A. Toumi. Rational verification: From model checking to equilibrium checking. In *Proc. of 30th National Conf. on Artificial Intelligence*, pages 4184–4190, 2016.

A Automata

For a set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow

the formulas **true** and **false**. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ makes θ true. An *alternating tree automaton* is $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$, where Σ is the input alphabet, D is a set of directions, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$ is a transition function, $q_{in} \in Q$ is an initial state, and $\alpha \subseteq Q$ specifies a Büchi or a co-Büchi acceptance condition. For a state $q \in Q$, we use $\mathcal{A}^q$ to denote the automaton obtained from $\mathcal{A}$ by setting the initial state to be q. The *size* of $\mathcal{A}$, denoted $|\mathcal{A}|$, is the sum of lengths of formulas that appear in δ.

The alternating automaton $\mathcal{A}$ runs on Σ-labeled D-trees. A *run* of $\mathcal{A}$ over a Σ-labeled D-tree $\langle T, \tau \rangle$ is a $(T \times Q)$-labeled $\mathbb{N}$-tree $\langle T_r, r \rangle$. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T. The labels of a node and its successors have to satisfy the transition function. Formally, $\langle T_r, r \rangle$ satisfies the following:

1. (1) $\varepsilon \in T_r$ and $r(\varepsilon) = \langle \varepsilon, q_{in} \rangle$.
2. (2) Let $y \in T_r$ with $r(y) = \langle x, q \rangle$ and $\delta(q, \tau(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_0, q_0), (c_1, q_1), \ldots, (c_{n-1}, q_{n-1})\} \subseteq D \times Q$, such that S satisfies θ, and for all $0 \leq i \leq n-1$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = \langle x \cdot c_i, q_i \rangle$.

For example, if $\langle T, \tau \rangle$ is a $\{0,1\}$-tree with $\tau(\varepsilon) = a$ and $\delta(q_{in}, a) = ((0, q_1) \vee (0, q_2)) \wedge ((0, q_3) \vee (1, q_2))$, then, at level 1, the run $\langle T_r, r \rangle$ includes a node labeled $(0, q_1)$ or a node labeled $(0, q_2)$, and includes a node labeled $(0, q_3)$ or a node labeled $(1, q_2)$. Note that if, for some y, the transition function δ has the value **true**, then y need not have successors. Also, δ can never have the value **false** in a run.

A run $\langle T_r, r \rangle$ is accepting if all its infinite paths satisfy the acceptance condition. Given a run $\langle T_r, r \rangle$ and an infinite path $\pi \subseteq T_r$, let $inf(\pi) \subseteq Q$ be such that $q \in inf(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in T \times \{q\}$. That is, $inf(\pi)$ contains exactly all the states that appear infinitely often in π. A path π satisfies a *Büchi* acceptance condition α iff $inf(\pi) \cap \alpha \neq \emptyset$, and satisfies a *co-Büchi* acceptance condition α iff $inf(\pi) \cap \alpha = \emptyset$. We also consider the *parity* acceptance condition, where $\alpha : Q \to \{0, 1, \ldots, k\}$ maps each state to a color in $\{0, 1, \ldots, k\}$, and a path π satisfies α if the minimal color visited infinitely often is even, thus $\min\{i : inf(\pi) \cap \alpha^{-1}(i) \neq \emptyset\}$ is even. An automaton accepts a tree iff there exists a run that accepts it. We denote by $L(\mathcal{A})$ the set of all Σ-labeled trees that $\mathcal{A}$ accepts.

The alternating automaton $\mathcal{A}$ is *nondeterministic* if for all the formulas that appear in δ, if (c_1, q_1) and (c_2, q_2) are conjunctively related, then $c_1 \neq c_2$. (i.e., if the transition is rewritten in disjunctive normal form, there is at most one element of $\{c\} \times Q$, for each $c \in D$, in each disjunct). The automaton $\mathcal{A}$ is *universal* if all the formulas that appear in δ are conjunctions of atoms in $D \times Q$, and $\mathcal{A}$ is *deterministic* if it is both nondeterministic and universal. The automaton $\mathcal{A}$ is a *word* automaton if $|D| = 1$. Then, we can omit D from the specification of

the automaton and denote the transition function of $\mathcal{A}$ as $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$. If the word automaton is nondeterministic or universal, then $\delta : Q \times \Sigma \to 2^Q$.

B Proofs

B.1 Proof of Theorem 1

We start with the NBT $\mathcal{N}_{\mathsf{W}}$. Recall that we want $\mathcal{N}_{\mathsf{W}}$ to accept a profile tree $\langle V^*, \pi \rangle$ iff exactly the players in W win in $\mathsf{Outcome}(\pi)$. Let $\psi = \bigwedge_{i \in \mathsf{W}} \psi_i \wedge \bigwedge_{i \notin \mathsf{W}} (\neg \psi_i)$, and let $\mathcal{A} = \langle 2^{AP}, Q, q_0, \mu, \alpha \rangle$ be an NBW of size exponential in $|\psi|$ that corresponds to ψ. The NBT $\mathcal{N}_{\mathsf{W}}$ follows the outcome in the profile tree, and checks if the players in W are exactly the winners of the profile. Formally, $\mathcal{N}_{\mathsf{W}} = \langle A_1 \times \cdots \times A_k, V, Q', q'_0, \eta, \alpha' \rangle$, where

1. $Q' = V \times Q$.
2. $q'_0 = \langle v_0, q_0 \rangle$.
3. For every $\langle v, q \rangle \in V \times Q$ and $\langle a_1, \ldots, a_k \rangle \in A_1 \times \cdots \times A_k$, we have that $\eta(\langle v, q \rangle, \langle a_1, \ldots, a_k \rangle) = \bigvee_{q' \in \mu(q, \tau(v))} (\delta(v, \langle a_1, \ldots, a_k \rangle), \langle \delta(v, \langle a_1, \ldots, a_k \rangle), q' \rangle)$.
4. $\alpha' = V \times \alpha$.

We continue to the UCT $\mathcal{U}^i_{\mathsf{W}}$. Recall that we want $\mathcal{U}^i_{\mathsf{W}}$ to accept a profile tree $\langle V^*, \pi \rangle$ iff Player i loses in $\mathsf{Outcome}(\pi[i \leftarrow f'_i])$, for every strategy f'_i. Let $\mathcal{U}_i = \langle 2^{AP}, Q_i, q^0_i, \delta_i, \alpha_i \rangle$ and $\neg \mathcal{U}_i = \langle 2^{AP}, S_i, s^0_i, \mu_i, \beta_i \rangle$ be the UCWs corresponding to ψ_i and $\neg \psi_i$, respectively. The UCT $\mathcal{U}^i_{\mathsf{W}}$ follows every possible deviation for Player i, and checks that indeed she always loses. Formally, $\mathcal{U}^i_{\mathsf{W}} = \langle A_1 \times \cdots \times A_k, V, Q, q_0, \eta, \alpha \rangle$, where

1. $Q = V \times S_i$.
2. $q_0 = \langle v_0, s^0_i \rangle$.
3. For every $\langle v, s \rangle \in V \times S_i$ and $\langle a_1, \ldots, a_k \rangle \in A_1 \times \cdots \times A_k$, we have that $\eta(\langle v, s \rangle, \langle a_1, \ldots, a_k \rangle) =$ $\bigwedge_{a'_i \in \kappa_i(v)} \bigwedge_{s' \in \mu_i(s, \tau(v))} (\delta(v, \langle a_1, \ldots, a'_i, \ldots, a_k \rangle), \langle \delta(v, \langle a_1, \ldots, a'_i, \ldots, a_k \rangle), s' \rangle)$.
4. $\alpha = V \times \beta_i$.

B.2 Proof of the EXPTIME lower bound in Theorem 2

We start with a definition of an alternating Turing machine (ATM). An ATM is a tuple $M = \langle Q_e, Q_u, \Gamma, \Delta, q_{init}, q_{acc}, q_{rej} \rangle$, where Γ is the alphabet, Q_e and Q_u are finite sets of *existential* and *universal* states, and we let $Q = Q_e \cup Q_u$. Then, q_{init}, q_{acc}, and q_{rej} are the initial, accepting, and rejecting states, respectively, and we assume that $q_{init} \in Q_e$. Finally, $\Delta \subseteq (Q \times \Gamma) \times ((Q \times \Gamma \times \{L, R\}) \times (Q \times \Gamma \times \{L, R\}))$ is a transition relation that in our case has a binary branching degree. When an existential or a universal state of M branches into two states, we distinguish between the left and right branches. Accordingly, we use $((q, \gamma), \langle (q_l, \gamma_l, d_l), (q_r, \gamma_r, d_r) \rangle)$ to indicate that when M is in state $q \in Q_e \cup Q_u$ reading input symbol γ, it branches to the left with (q_l, γ_l, d_l) and to the right

with (q_r, γ_r, d_r). Note that directions left and right here have nothing to do with the movement direction of the head. These are determined by d_l and d_r.

A configuration of M on $w = w_1, \ldots, w_n$ describes its state, the content of the working tape, and the location of the reading head. Assume $s : \mathbb{N} \to \mathbb{N}$ is a linear function such that the number of cells used by the working tape in every configuration of M on its run on w is bounded by $s(n)$. We encode a configuration of M by a string $\#\gamma_1\gamma_2 \cdots (q, \gamma_i) \cdots \gamma_{s(n)}$. That is, a configuration starts with $\#$, and all its other letters are in Γ, except for one letter in $Q \times \Gamma$. Then, M is in state q, the content of the j-th tape cell is γ_j, and the reading head points to cell i. We say that the configuration is *existential* if $q \in Q_e$ and that it is *universal* if $q \in Q_u$. The initial configuration of M on w, is then $\#(q_{init}, w_1) \cdot \ldots \cdot w_n \cdot \sqcup^{s(n)-n}$, for the special letter $\sqcup \in \Gamma$. We also assume that the initial configuration is existential. If the current state is q_{acc} or q_{rej}, then the configuration is final and has no successors. Otherwise, the left and right successors of a configuration are determined by Δ.

For a configuration c of M, let $succ_l(c)$ and $succ_r(c)$ be the successors of c when applying to it the left and right choices in Δ, respectively. Given an input word w, a computation tree of M on w is a tree in which each node corresponds to a configuration of M. The root of the tree corresponds to the initial configuration. A node that corresponds to a universal configuration c has two successors, corresponding to $succ_l(c)$ and $succ_r(c)$. A node that corresponds to an existential configuration c has a single successor, corresponding to either $succ_l(c)$ or $succ_r(c)$. The tree is an accepting computation tree if all its branches reach an accepting configuration. We can now encode a branch of the computation tree of M by a sequence of configurations.

In the membership problem, we get as input an ATM M and a word $w \in \Gamma^*$, and we decide whether M accepts w. The membership problem is EXPTIME-hard already for M of a fixed size, and when Δ alternates between existential and universal states, thus $\Delta \subseteq (Q_e \times \Gamma \times Q_u \times \Gamma \times \{L, R\}) \cup (Q_u \times \Gamma \times Q_e \times \Gamma \times \{L, R\})$. So for simplicity, we assume that M behaves this way.

We now proceed to the reduction from the membership problem for a linear-space ATM, which is known to be EXPTIME-complete [5], to CRS with G and ψ_2 are of a fixed size. That is, given an ATM M with space complexity $s : \mathbb{N} \to \mathbb{N}$ and a word w, we construct a 2-player turn-based game $\mathcal{G} = \langle G, \{\psi_1, \psi_2\}\rangle$, such that G and ψ_2 are of a fixed size, ψ_1 is of size linear in $s(|w|)$, and there is a CRS solution in $\mathcal{G}$ iff M accepts w.

Let $n = |w|$. Essentially, Player 1 generates a legal computation in the computation tree of M on w by choosing the letters of the current configuration one by one. Also, at the end of every existential configuration, Player 1 chooses whether to continue to the left or right successor configuration by choosing l or r, respectively. Player 2, on the other hand, only chooses the direction of the successor configuration after every universal configuration. The play induces a sequence that alternates between tape cell content and branching choices, thus a sequence of the form $\ldots \#d\gamma_1\gamma_2 \ldots (q, \gamma_i) \ldots \gamma_{s(n)} \#d'\gamma'_1\gamma'_2 \ldots (q', \gamma'_{i'}) \ldots \gamma'_{s(n)} \ldots$, where $d, d' \in \{l, r\}$. Note that the play describes both a sequence of consecutive

configurations of M and a branch in the computation tree of M on w. Then, the objective of Player 1 is to reach an accepting configuration, and the objective of Player 2 is to reach a rejecting configuration.

The challenge is to force Player 1 to construct a correct branch in the computation tree , and to do it with a game graph and objective for Player 2 of a fixed size. To do that, we first describe the function $next_l$ (the function $next_r$ is defined the same way for the right branch); Let $\Sigma = \{\#\} \cup (Q \times \Gamma) \cup \Gamma$ and let $\#\sigma_1 \dots \sigma_{s(n)} \# \sigma'_1 \dots \sigma'_{s(n)}$ be two successive configurations c and $succ_l(c)$ of M. We also set σ_0, σ'_0 and $\sigma_{s(n)+1}$ to $\#$. For each triple $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$, with $1 \leq i \leq n$, we know, by the transition relation of M, what σ'_i should be. In addition, the letter $\#$ should repeat exactly every $s(n)+1$ letters. Let $next_l(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$ denote our expectation for σ'_i in $succ_l(c)$. That is:

1. $next_l(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle) = next_l(\langle \#, \sigma_i, \sigma_{i+1} \rangle) = next_l(\langle \sigma_{i-1}, \sigma_i, \# \rangle) = \sigma_i$.
2. $next_l(\langle (q, \sigma_{i-1}), \sigma_i, \sigma_{i+1} \rangle) = next_l(\langle (q, \sigma_{i-1}), \sigma_i, \# \rangle) =$
   $\begin{cases} \sigma_i & \text{If } ((q, \sigma_{i-1}), \langle (q', \sigma'_{i-1}, L), (q_r, \sigma_r, d_r) \rangle) \in \Delta \\ (q', \sigma_i) & \text{If } ((q, \sigma_{i-1}), \langle (q', \sigma'_{i-1}, R), (q_r, \sigma_r, d_r) \rangle) \in \Delta \end{cases}$
3. $next_l(\langle \sigma_{i-1}, (q, \sigma_i), \sigma_{i+1} \rangle) = next_l(\langle \#, (q, \sigma_i), \sigma_{i+1} \rangle) = next_l(\langle \sigma_{i-1}, (q, \sigma_i), \# \rangle) = \sigma'_i$, where $((q, \sigma_i), \langle (q', \sigma'_i, d), (q_r, \sigma_r, d_r) \rangle) \in \Delta$.
4. $next_l(\langle \sigma_{i-1}, \sigma_i, (q, \sigma_{i+1}) \rangle) = next_l(\langle (\#, \sigma_i, (q, \sigma_{i+1}) \rangle) =$
   $\begin{cases} \sigma_i & \text{If } ((q, \sigma_{i-1}), \langle (q', \sigma'_{i+1}, R), (q_r, \sigma_r, d_r) \rangle) \in \Delta \\ (q', \sigma_i) & \text{If } ((q, \sigma_{i-1}), \langle (q', \sigma'_{i+1}, L), (q_r, \sigma_r, d_r) \rangle) \in \Delta \end{cases}$
5. $next_l(\langle \sigma_{s(n)}, \#, \sigma'_1 \rangle) = \#$.

Consistency with $next_l$ and $next_r$ now gives us a necessary condition for a trace to encode a legal branch of a computation tree.

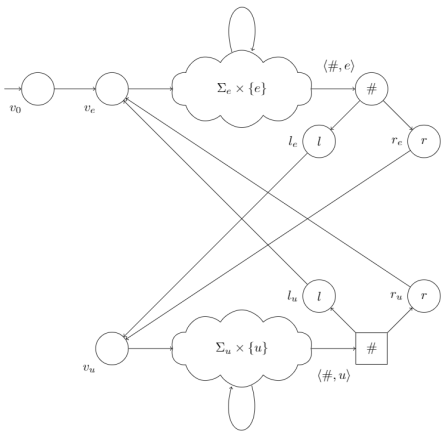

Fig. 1. The game graph G. The circles are vertices controlled by Player 1, and the squares are vertices controlled by Player 2.

Now we specify the formal definitions of G, ψ_1 and ψ_2.

1. The game graph $G = \langle AP, V_1, V_2, v_0, E, \tau \rangle$ is defined as follows (see Figure 1):
   (a) $AP = \Sigma \cup \{l, r\}$. We use $\Sigma_e = (Q_e \times \Gamma) \cup \Gamma$ and $\Sigma_u = (Q_u \times \Gamma) \cup \Gamma$.
   (b) $V_1 = \{v_0\} \cup \bigcup_{t \in \{e,u\}} \{l_t, r_t, v_t\} \cup ((\Sigma_e \cup \{\#\}) \times \{e\}) \cup (\Sigma_u \times \{u\})$.
   The vertex v_0 is the initial vertex. The vertices $\Sigma_e \times \{e\}$ are *the existential content vertices*, which are used to form the existential configurations. The same way, $\Sigma_u \times \{u\}$ are *the universal content vertices*.
   The vertices l_e, r_e, l_u and r_u represent the branching choices. At the end of an existential configuration, Player 1 chooses what direction to proceed to by moving to l_e or r_e from $\langle \#, e \rangle$, and at the end of an universal configuration, Player 2 makes that choice at the vertex $\langle \#, u \rangle$, by choosing either l_u or r_u. From both l_e and r_e, Player 1 moves to v_u, to start the successor universal configuration. The same way, from both l_u and r_u, Player 1 moves to v_e, to start the successor existential configuration.
   (c) $V_2 = \{\langle \#, u \rangle\}$. the vertex $\langle \#, u \rangle$ is the vertex that represent the end of an universal configuration, and upon arriving to it, Player 2 chooses what direction to proceed to by moving to l_u or r_u.
   (d) The set E contains the following edges:
      i. $\langle v_0, v_e \rangle$.
      ii. For every $t \in \{e, u\}$ we have the following edges:
         - $\langle v_t, \langle \sigma, t \rangle \rangle$ for every $\sigma \in (Q_t \times \Gamma) \cup \Gamma \cup \{\#\}$.
         - $\langle \langle \sigma, t \rangle, \langle \sigma', t \rangle \rangle$ for every $\sigma \in (Q_t \times \Gamma) \cup \Gamma$, $\sigma' \in (Q_t \times \Gamma) \cup \Gamma \cup \{\#\}$.
         - $\langle \langle \#, t \rangle, l_t \rangle$ and $\langle \langle \#, t \rangle, r_t \rangle$.
         - $\langle l_t, v_{t'} \rangle$ and $\langle r_t, v_{t'} \rangle$ where $t' = \{e, u\} \setminus \{t\}$.
   (e) The labeling of the vertices is as follows:
      i. $\tau(v) = \emptyset$, for every $v \in \{v_0\} \cup \bigcup_{t \in \{e,u\}} \{v_t\}$.
      ii. $\tau(\langle \sigma, t \rangle) = \sigma$, for every $\sigma \in (Q_t \times \Gamma) \cup \Gamma \cup \{\#\}$ and $t \in \{e, u\}$.
      iii. $\tau(v) = l$, for every $v \in \{l_e, l_u\}$.
      iv. $\tau(v) = r$, for every $v \in \{r_e, r_u\}$.
2. The objective of Player 2 is given by the LTL formula $\psi_2 = \bigvee_{\gamma \in \Gamma} F(q_{rej}, \gamma)$.
3. The objective of Player 1 is the LTL formula $\psi_1 = \psi_{init} \wedge \psi_{next_l} \wedge \psi_{next_r} \wedge \psi_{acc}$, with the following conjuncts:
   (a) The computation starts with the initial configuration: $\psi_{init} = \# \wedge X((q_{init}, w_1) \wedge X(w_2 \wedge X(\cdots w_n \wedge X(\sqcup \wedge X(\cdots \sqcup)))))$.
   (b) The computation is consistent with $next_l$ between the same tape cell in consecutive configurations with the l branching choice between them: $\psi_{next_l} = \bigwedge_{\sigma_1, \sigma_2, \sigma_3 \in \Sigma^3} G((\sigma_1 \wedge X(\sigma_2 \wedge X(\sigma_3)) \wedge (\neg\#)U(\# \wedge Xl)) \rightarrow X^{s(n)+2} next_l(\langle \sigma_1, \sigma_2, \sigma_3 \rangle))$.
   (c) The computation is consistent with $next_r$ between the same tape cell in consecutive configurations with the r branching choice between them: $\psi_{next_r} = \bigwedge_{\sigma_1, \sigma_2, \sigma_3 \in \Sigma^3} G((\sigma_1 \wedge X(\sigma_2 \wedge X(\sigma_3)) \wedge (\neg\#)U(\# \wedge Xr)) \rightarrow X^{s(n)+2} next_r(\langle \sigma_1, \sigma_2, \sigma_3 \rangle))$.
   (d) The computation reaches an accepting configuration: $\psi_{acc} = \bigvee_{\gamma \in \Gamma} F(q_{acc}, \gamma)$.

The sizes of G and ψ_2 are fixed because the size of M is fixed, and the size of ψ_1 is linear in $s(n)$.

B.3 Proof of the 2EXPTIME lower bound in Theorem 2

The game graph $H = \langle AP', V', v'_0, \{A'_1, A'_2\}, \{\kappa'_1, \kappa'_2\}, \delta', \tau' \rangle$ is defined as follows (see Figure 2):

1. $AP' = AP \cup \{p\}$, where $p \notin AP$ is a new atomic proposition that is used to distinguish between the two copies of G.
2. $V' = \{v'_0\} \cup (V \times \{1, 2\})$. The vertex v'_0 is the new initial vertex in which Player 2 chooses the copy to proceed to.
3. $A'_1 = A_1$.
4. $A'_2 = A_2 \cup \{c_1, c_2\}$. We add two actions for Player 2, for the purpose of choosing between the copies. In the initial vertex, Player 2 can either choose the action c_1, and then the game proceeds to the first copy, or choose the action c_2, and then the game proceeds to the second copy.
5. For $j \in \{1, 2\}$ and $v \in V$, we have that $\kappa'_1(\langle v, j \rangle) = \kappa_1(v)$, and $\kappa'_1(v'_0) = A'_1$.
6. For $j \in \{1, 2\}$ and $v \in V$, we have that $\kappa'_2(\langle v, j \rangle) = \kappa_2(v)$, and $\kappa'_2(v'_0) = \{c_1, c_2\}$.
7. The transition function δ' is defined as follows:
   (a) $\delta'(v'_0, \langle a_1, c_1 \rangle) = \langle v_0, 1 \rangle$, for every $a_1 \in A'_1$.
   (b) $\delta'(v'_0, \langle a_1, c_2 \rangle) = \langle v_0, 2 \rangle$, for every $a_1 \in A'_1$.
   (c) $\delta'(\langle v, j \rangle, \langle a_1, a_2 \rangle) = \langle \delta(v, \langle a_1, a_2 \rangle), j \rangle$, for every $j \in \{1, 2\}$, $v \in V$, $a_1 \in \kappa'_1(\langle v, j \rangle)$, and $a_2 \in \kappa'_2(\langle v, j \rangle)$.
8. The labeling of the vertices is as follows:
   (a) $\tau'(v'_0) = \emptyset$.
   (b) $\tau'(v) = \tau(v) \cup \{p\}$, for every $v \in V \times \{1\}$.
   (c) $\tau'(v) = \tau(v)$, for every $v \in V \times \{2\}$.

The objective of Player 1 is given by the LTL formula $\psi_1 = XGp$, and the objective of Player 2 is given by the LTL formula $\psi_2 = X(\neg\psi \wedge G\neg p)$.

The size of H is linear in $|G|$, the size of ψ_1 is fixed, and the size of ψ_2 is linear in $|\psi|$. Then, the size of H if fixed when the size of G is fixed, and as deciding zero-sum games is 2EXPTIME complete already for a fixed size graph, the above complexity holds.

If Player 1 wins $\mathcal{G}$, then she has a winning strategy f_1 that ensures the satisfaction of ψ for all possible behaviors of Player 2. Let $\pi = \langle f'_1, f_2 \rangle$ be a profile in $\mathcal{H}$, where f'_1 follows the winning strategy f_1 in the copy that Player 2 chose, and f_2 is some strategy for Player 2 that chooses the first copy. Then, the play is generated in the first copy, and the induced computation satisfies ψ. It is easy to see that π is a $\{1\}$-NE in $\mathcal{H}$. Indeed, Player 1 wins, as f'_1 ensures the satisfaction of ψ for every possible strategy of Player 2, and there is no beneficial deviation for Player 2.

If Player 1 loses $\mathcal{G}$, she has no winning strategy in $\mathcal{G}$, and so for every strategy f_1 for Player 1, there is a strategy f_2 for Player 2 such that $\mathsf{Outcome}(f_1, f_2)$ does not satisfy ψ. Then, there is no $\{1\}$-NE in $\mathcal{H}$. Indeed, Player 2 always has a beneficial deviation, in the form of a strategy that both chooses the second copy, and force the computation to not satisfy ψ.

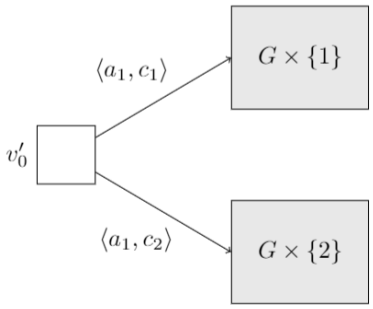

Fig. 2. The game graph H. Player 2 chooses between the actions c_1 and c_2 at the initial vertex v'_0, and then the game proceed to the corresponding copy of G.

B.4 Proof of Theorem 4

We define a UCT $\mathcal{U}$ that accepts a $((V \cup \{\oslash\}) \times [k])$-labeled V-tree iff it is a certified strategy tree. Let $\mathcal{U}_i = \langle 2^{AP}, Q_i, q_i^0, \delta_i, \alpha_i \rangle$ and $\neg\mathcal{U}_i = \langle 2^{AP}, S_i, s_i^0, \mu_i, \beta_i \rangle$ be the UCWs corresponding to ψ_i and $\neg\psi_i$, respectively. We define $\mathcal{U}$ as the intersection of the following UCTs.

1. For every $i \in [k] \setminus \{1\}$, a UCT $\mathcal{C}_1^i$ that checks the satisfaction of $(\mathbf{C_1^i})$. The size of $\mathcal{C}_1^i$ is polynomial in $|G|$ and $|\mathcal{U}_i|$.
2. A UCT $\mathcal{C}_2$ that checks the satisfaction of $(\mathbf{C_2})$. The size of $\mathcal{C}_2$ is polynomial in $|G|$.
3. For every $i \in [k] \setminus \{1\}$, A UCT $\mathcal{C}_3^i$ that checks the satisfaction of $(\mathbf{C_3^i})$. The size of $\mathcal{C}_3^i$ is polynomial in $|G|$.
4. For every $i \in [k]$, A UCT $\mathcal{C}_4^i$ that checks the satisfaction of $(\mathbf{C_4^i})$. The size of C_4^1 is polynomial in $|G|$ and $|\mathcal{U}_1|$, and for every $i \in [k] \setminus \{1\}$, the size of C_4^i is polynomial in $|G|$ and in $\neg\mathcal{U}_i$.

By the above, the UCT $\mathcal{U}$ is polynomial in $|G|$ and in the UCWs for $\psi_1, \psi_2, \neg\psi_2, \ldots, \psi_k, \neg\psi_k$. Hence, it is polynomial in $|G|$ and exponential in $|\psi_1|, |\psi_2|, \ldots, |\psi_k|$.

Below we describe the construction of the underlying UCTs.

$\mathcal{C}_1^i$: a UCT that checks $(\mathbf{C_1^i})$ We construct a UCT $\mathcal{C}_1^i$ that checks the deviations of Player i in the certified strategy tree. In order to do so, the UCT traverses the tree, and checks for sub-trees containing deviations for Player i. The root of such a sub-tree is a node owned by Player i that is s-labeled by some successor, rather than $\oslash$. Then, for every such sub-tree, the UCT checks that the deviation f_i is indeed a winning strategy for Player i, by verifying the satisfaction of ψ_i in paths that agree with f_1 and the deviation for Player i in the sub-tree.

We define $\mathcal{C}_1^i = \langle (V \cup \{\oslash\}) \times [k], V, Q, q_0, \delta, \alpha \rangle$, where

1. $Q = V \times Q_i \times \{in, out\}$. The in/out flag indicates whether we are in a sub-tree that specifies a deviation for Player i or not.

2. $q_0 = \langle v_0, q_i^0, out \rangle$.
3. For all $\langle v, q, in \rangle \in V \times Q_i \times \{in\}$ and $\langle v', l \rangle \in (V \cup \{\oslash\}) \times [k]$, the transition function δ is defined as follows:
   (a) If $v \in V_1 \cup V_i$, then $\mathcal{C}_1^i$ reads a sub-tree that contains a deviation for Player i, and it reads a node that is s-labeled by the strategy of Player 1 or by the deviation of Player i. Accordingly, $\delta(\langle v, q, in \rangle, \langle v', l \rangle) = \bigwedge_{q' \in \delta_i(q, \tau(v))} \langle v', \langle v', q', in \rangle \rangle$.
   (b) Otherwise, $\mathcal{C}_1^i$ reads a sub-tree that contains a deviation for Player i, and it reads a node owned by one of the other players. Accordingly, $\delta(\langle v, q, in \rangle, \langle v', l \rangle) = \bigwedge_{\{v'' : (v, v'') \in E\}} \bigwedge_{q' \in \delta_i(q, \tau(v))} \langle v'', \langle v'', q', in \rangle \rangle$.
4. For all $\langle v, q, out \rangle \in V \times Q_i \times \{out\}$ and $\langle v', l \rangle \in (V \cup \{\oslash\}) \times [k]$, the transition function δ is defined as follows:
   (a) If $v \in V_1$, then $\delta(\langle v, q, out \rangle, \langle v', l \rangle) = \bigwedge_{q' \in \delta_i(q, \tau(v))} \langle v', \langle v', q', out \rangle \rangle$.
   (b) If $v \in V_j$ for $j \in [k] \setminus \{1, i\}$, then $\delta(\langle v, q, out \rangle, \langle v', l \rangle) = \bigwedge_{\{v'' : (v, v'') \in E\}} \bigwedge_{q' \in \delta_i(q, \tau(v))} \langle v'', \langle v'', q', out \rangle \rangle$.
   (c) If $v \in V_i$, the UCT has not yet entered a sub-tree that specifies a deviation for Player i. Upon arriving to a node owned by Player i, it might be still outside a deviation, or declares a beginning of a deviation. In the first case, the node is not s-labeled by a successor of v. In the second case, the node v is s-labeled by a possible successor, and the UCT moves to an "in" mode:
      i. If $v' = \oslash$, then $\delta(\langle v, q, out \rangle, \langle v', l \rangle) = \bigwedge_{\{v'' : (v, v'') \in E\}} \bigwedge_{q' \in \delta_i(q, \tau(v))} \langle v'', \langle v'', q', out \rangle \rangle$.
      ii. If $v' \in V$, then $\delta(\langle v, q, out \rangle, \langle v', l \rangle) = \bigwedge_{q' \in \delta_i(q, \tau(v))} \langle v', \langle v', q', in \rangle \rangle$.
5. $\alpha = V \times \alpha_i \times \{in\}$. Note that only paths in sub-trees that specify deviations are required to satisfy the α_i acceptance condition.

$\mathcal{C}_2$: a DCT that checks ($\mathbf{C_2}$) We construct a DCT $\mathcal{C}_2$ that for every path in the certified tree that is consistent with f_1, checks that the suffix of the path is p-labeled by a single $i \in [k]$. We define $\mathcal{C}_2 = \langle (V \cup \{\oslash\}) \times [k], V, Q, q_0, \delta, \alpha \rangle$, where

1. $Q = V \times [k] \times \{Y, N\}$. The $[k]$ component indicates the expected p-label. The Y/N flag indicates whether the p-label of the current node is consistent with the p-label of the predecessor node, or not. Our goal is to have a finite number of instances of such inconsistencies in each path. If there is a finite number of inconsistencies, we know that for each path, the suffix is p-labeled by a single $i \in [k]$.
2. $q_0 = \langle v_0, 1, Y \rangle$.
3. For all $\langle v, i, flag \rangle \in V \times [k] \times \{Y, N\}$ and $v' \in (V \cup \{\oslash\})$, the transition function δ is defined as follows:
   (a) If $v \in V_1$:
      i. $\delta(\langle v, i, flag \rangle, \langle v', i \rangle) = \langle v', \langle v', i, Y \rangle \rangle$
      ii. $\delta(\langle v, i, flag \rangle, \langle v', j \rangle) = \langle v', \langle v', j, N \rangle \rangle$, for every $j \in [k] \setminus \{i\}$.
   (b) Otherwise:

i. $\delta(\langle v, i, flag\rangle, \langle v', i\rangle) = \bigwedge_{\{v'':(v,v'')\in E\}} \langle v'', \langle v'', i, Y\rangle\rangle$.
ii. $\delta(\langle v, i, flag\rangle, \langle v', j\rangle) = \bigwedge_{\{v'':(v,v'')\in E\}} \langle v'', \langle v'', j, N\rangle\rangle$, for every $j \in [k] \setminus \{i\}$.

4. $\alpha = V \times [k] \times \{N\}$.

$\mathcal{C}_3^i$: a DCT that checks ($\mathbf{C_3^i}$) We construct a DCT $\mathcal{C}_3^i$ that checks that for every node x that is p-labeled by i, there is a good deviation point for Player i in the path from the root to x. That is, that there was a node corresponding to a vertex controlled by Player i, s-labeled by a possible successor. We define $\mathcal{C}_3^i = \langle (V \cup \{\oslash\}) \times [k], V, Q, q_0, \delta, \alpha\rangle$, where

1. $Q = V$. When $\mathcal{C}_3^i$ sees a node p-labeled by i without detecting a good deviation point for Player i earlier, it rejects the tree.
2. $q_0 = v_0$.
3. For all $v \in V$, the transition function δ is defined as follows:
   (a) $\delta(v, \langle v', i\rangle) = \mathbf{false}$.
   (b) If $v \in V_1$, then $\delta(v, \langle v', j\rangle) = \langle v', v'\rangle$ for every $j \in [k] \setminus \{i\}$.
   (c) If $v \in V_i$, then $\delta(v, \langle v', j\rangle) = \mathbf{true}$, for every $j \in [k] \setminus \{i\}$ and $v' \in V$ such that $(v, v') \in E$, and $\delta(v, \langle \oslash, j\rangle) = \bigwedge_{\{v'':(v,v'')\in E\}} \langle v'', v''\rangle$ for every $j \in [k] \setminus \{i\}$.
   (d) Otherwise, $\delta(v, \langle v', j\rangle) = \bigwedge_{\{v'':(v,v'')\in E\}} \langle v'', v''\rangle$ for every $j \in [k] \setminus \{i\}$.
4. $\alpha = \emptyset$.

$\mathcal{C}_4^i$: a UCT that checks ($\mathbf{C_4^i}$) We construct a UCT $\mathcal{C}_4^i$ that checks the correctness of the p-labeling regarding Player i. That is, C_4^1 checks that Player 1 wins in every path that its suffix is p-labeled by 1, and for every $i \in [k] \setminus \{1\}$, C_4^i checks that Player i loses in every path that its suffix is p-labeled by i. Thus, every path whose suffix is p-labeled by i satisfies $\neg\psi_i$. Here we define C_4^i for $i \in [k] \setminus \{1\}$. The definition for C_4^1 is very similar, with the difference of using $\mathcal{U}_1$ in the construction, instead of $\neg\mathcal{U}_1$.

We define $\mathcal{C}_4^i = \langle (V \cup \{\oslash\}) \times [k], V, S, s_0, \mu, \beta\rangle$, where

1. $S = V \times S_i \times \{Y, N\}$. The automaton follows the current vertex of the game, the state in the UCW corresponding to $\neg\psi_i$, and p-labeling of the nodes in the tree. The Y/N flag indicates whether the current p-labeling is consistent with i.
2. $s_0 = \langle v_0, s_i^0, Y\rangle$.
3. For every $\langle v, s, c\rangle \in S$ and $\langle v', i\rangle \in (V \cup \{\oslash\}) \times \{i\}$, the transition function μ is defined as follows:
   (a) If $v \in V_1$, then $\mu(\langle v, s, c\rangle, \langle v', i\rangle) = \bigwedge_{s' \in \mu_i(s,\tau(v))} \langle v', \langle v', s', Y\rangle\rangle$.
   (b) Otherwise, $\mu(\langle v, s, c\rangle, \langle v', i\rangle) = \bigwedge_{\{v'':(v,v'')\in E\}} \bigwedge_{s' \in \mu_i(s,\tau(v))} \langle v'', \langle v'', s', Y\rangle\rangle$.
4. For every $\langle v, s, c\rangle \in S$ and $\langle v', j\rangle \in (V \cup \{\oslash\}) \times ([k] \setminus \{i\})$, the transition function μ is defined as follows:
   (a) If $v \in V_1$, then $\mu(\langle v, s, c\rangle, \langle v', j\rangle) = \bigwedge_{s' \in \mu_i(s,\tau(v))} \langle v', \langle v', s', N\rangle\rangle$.
   (b) Otherwise, $\mu(\langle v, s, c\rangle, \langle v', j\rangle) = \bigwedge_{\{v'':(v,v'')\in E\}} \bigwedge_{s' \in \mu_i(s,\tau(v))} \langle v'', \langle v'', s', N\rangle\rangle$.
5. $\beta = V \times \beta_i \times \{Y\}$. When the suffix of a path in the tree is p-labeled by i and it does not satisfy $\neg\psi_i$, then $\mathcal{C}_4^i$ visits infinitely many states in $V \times \beta_i \times \{Y\}$ in its run along this path. In this case, the UCT rejects the tree as it should.

B.5 Proof of Theorem 7

To simplify the explanation, we omit the suffix-labeling component (a.k.a., the p-labeling) from the description. Note that indeed the good deviation transitions labeling is only dependent on the s and d-labeling in the nodes. For a vertex v and an action $a_1 \in A$, let V_{v,a_1} be the set of vertices that can be successors of v, given that Player 1 chooses the action a_1. That is, $V_{v,a_1} = \{u \in V : \exists \langle a_2, \ldots, a_k\rangle \in A^{k-1} \text{ s.t. } \delta(v, \langle a_1, a_2, \ldots, a_k\rangle) = u\}$. Now we can define a UCT $\mathcal{U}$ over $(A_1 \times \mathcal{D})$-labeled V-trees such that $\mathcal{U}$ accepts a certified strategy tree $\langle V^*, g\rangle$ iff every good deviation transition annotation is correct. That is, if a node $h \cdot v$ is s-labeled by a_1, d-labeled by d, and there is a vector of actions $\langle a_2, \ldots, a_k\rangle$ such that $(d(\langle a_2, \ldots, a_k\rangle))_i \in A$, then $\delta(v, \langle a_1, a_2, \ldots, (d(\langle a_2, \ldots, a_k\rangle))_i, \ldots, a_k\rangle)$ is a winning point for Player i.

We define $\mathcal{U} = \langle A_1 \times \mathcal{D}, V, Q, q_0, \eta, \alpha\rangle$, where

1. $Q = V \times (Q_2 \times \cdots \times Q_k \cup (Q_2 \cup \cdots \cup Q_k)) \times (\{e\} \cup \bigcup_{i\in[k]\setminus\{1\}} v_i)$. The flag e is standing for *explore*, while for every $i \in \{2, ..., k\}$, the flag v_i is standing for *verifying* i. The $\{e, v_2, \ldots, v_k\}$ flags indicate whether the UCT is searching for good deviation transitions, or that the automaton already found such marking, and now it verifies the deviations.
2. $q_0 = \langle v_0, q_2^0, \ldots, q_k^0, e\rangle$.
3. The transition function η regarding the exploring is defined as follows:
   (a) For every $\langle v, q_2, \ldots, q_k, e\rangle$ and $\langle a_1, d\rangle \in A_1^v \times \mathcal{D}$, we have that
   $\eta(\langle v, q_2, \ldots, q_k, e\rangle, \langle a_1, d\rangle) =$
   $(\bigwedge_{u\in V_{v,a_1}} \bigwedge_{\overline{q}\in\delta_2(q_2,\tau(v))\times\cdots\times\delta_k(q_k,\tau(v))} (u, \langle u, \overline{q}, e\rangle)) \wedge$
   $(\bigwedge_{\langle a_2,\ldots,a_k\rangle\in A^{k-1}} \bigwedge_{\{i\in[k]\setminus\{1\}:(d(\langle a_2,\ldots,a_k\rangle))_i\in A\}} \bigwedge_{q_i'\in\delta_i(q_i,\tau(v))}$
   $(\delta(v, \langle a_1, a_2, \ldots, a_{i-1}, (d(\langle a_2, \ldots, a_k\rangle))_i, a_{i+1}, \ldots, a_k\rangle),$
   $\langle\delta(v, \langle a_1, a_2, \ldots, a_{i-1}, (d(\langle a_2, \ldots, a_k\rangle))_i, a_{i+1}, \ldots, a_k\rangle), q_i', v_i\rangle)).$
   The first part of the conjunction makes sure the UCT goes over the entire certified tree, and the second part verifies the deviations the UCT finds.
4. The transition function η is defined as follows:
   (a) For every $\langle v, q_i, v_i\rangle$ and $\langle a_1, d\rangle \in A_1^v \times \mathcal{D}$ such that for every $\langle a_2, \ldots, a_k\rangle \in A^{k-1}$ we have that $(d(\langle a_2, \ldots, a_k\rangle))_i \in A$, we have that
   $\eta(\langle v, q_i, v_i\rangle, \langle a_1, d\rangle) = \bigwedge_{\langle a_2,\ldots,a_k\rangle\in A^{k-1}} \bigwedge_{q_i'\in\delta_i(q_i,\tau(v))}$
   $(\delta(v, \langle a_1, a_2, \ldots., a_{i-1}, (d(\langle a_2, \ldots, a_k\rangle))_i, a_{i+1}, \ldots, a_k\rangle),$
   $\langle\delta(v, \langle a_1, a_2, \ldots., a_{i-1}, (d(\langle a_2, \ldots, a_k\rangle))_i, a_{i+1}, \ldots, a_k\rangle), q_i', v_i\rangle).$
5. $\alpha = \bigcup_{i\in[k]\setminus\{1\}} V \times \alpha_i \times \{v_i\}$. That is, we want to ensure that once we initially found a deviation for Player i, she can always has a beneficial deviation.

B.6 UCT for verifying good deviation transitions with vertical annotation

We define a UCT $\mathcal{U}$ over Σ-labeled $(V \cup T)$-trees such that $\mathcal{U}$ accepts a certified strategy tree $\langle (V \cup T)^*, f_1\rangle$ iff every good deviation transition annotation is correct. That is, if v, v' is marked as a good deviation transition for L, then there are a beneficial deviations for the players that deviate.

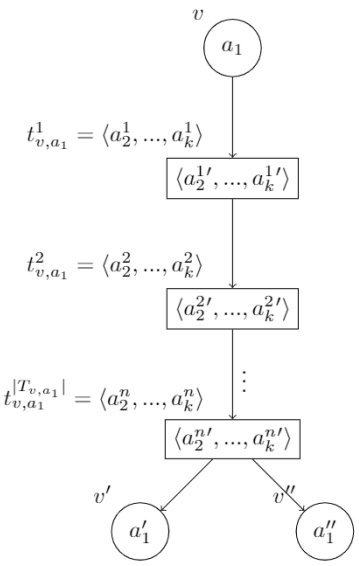

Fig. 3. A vertically certified strategy tree. Information about deviations from a given vector of actions is stored in an intermediate node that corresponds to that vector. There is no good deviation transition starting from v.

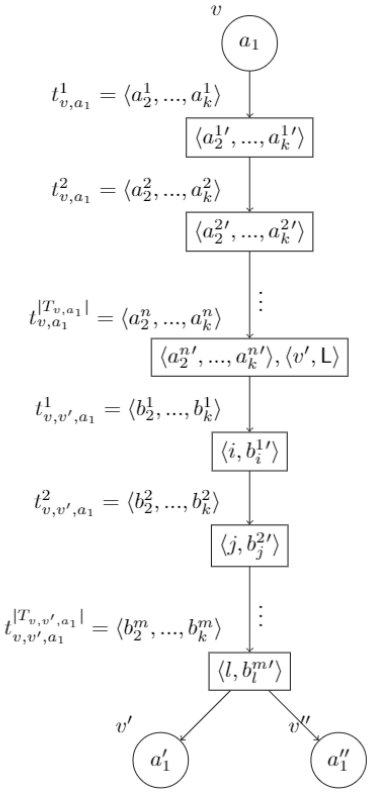

Fig. 4. v, v' is a good deviation transition for L.

The UCT $\mathcal{U}$ is the conjunction of $\mathcal{U}_{\mathsf{L}}$, where $\mathsf{L} \in 2^{[k]\setminus\{1\}} \setminus \emptyset$. We define $\mathcal{U}_{\mathsf{L}} = \langle \Sigma, V \cup T, Q, q_0, \eta, \alpha \rangle$, where

1. $Q = (V \cup T \cup (V \times T)) \times Q_2 \times \cdots \times Q_k \times [k] \times \{e, v\}$. The flags e and v stand for *explore* and *verify*, respectively. The $\{e, v\}$ flags indicate whether the automaton is searching for good deviation transitions for L, or that the automaton already found such a marking, and now it verifies the deviations.
2. $q_0 = \langle v, q^0_2, \ldots, q^0_k, 1, e \rangle$.
3. The transition function η regarding the exploring is defined as follows:
   (a) For every $\langle v, q_2, \ldots, q_k, 1, e \rangle$, and $a_1 \in \kappa_1(v)$, we have that $\eta(\langle v, q_2, \ldots, q_k, 1, e \rangle, a_1) = \bigwedge_{\overline{q} \in \delta_2(q_2, \tau(v)) \times \cdots \times \delta_k(q_k, \tau(v))} (t^1_{v,a_1}, \langle t^1_{v,a_1}, \overline{q}, 1, e \rangle)$.
   (b) For every $\langle t^i_{v,a_1}, \overline{q}, 1, e \rangle$, $1 \leq i \leq |T_{v,a_1}| - 1$ and $\sigma \in \Sigma$, we have that $\eta(\langle t^i_{v,a_1}, \overline{q}, 1, e \rangle, \sigma) = (t^{i+1}_{v,a_1}, \langle t^{i+1}_{v,a_1}, \overline{q}, 1, e \rangle)$.
   (c) For every $\langle t^i_{v,v',a_1}, \overline{q}, 1, e \rangle$, $1 \leq i \leq |T_{v,v',a_1}| - 1$ and $\sigma \in \Sigma$, we have that $\eta(\langle t^i_{v,v',a_1}, \overline{q}, 1, e \rangle, \sigma) = (t^{i+1}_{v,v',a_1}, \langle t^{i+1}_{v,v',a_1}, \overline{q}, 1, e \rangle)$.

(d) For every $\langle t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle$, $\langle t_{v,v',a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle$ and $\overline{a}$, we have that
$\eta(\langle t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle, \overline{a}) = \eta(\langle t_{v,v',a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle, \overline{a}) = \bigwedge_{\{u:(v,u)\in E\}}(u, \langle u, \overline{q}, 1, e\rangle)$.

(e) For every $\langle t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle$ and $\langle \overline{a}, \langle v', \mathsf{L}\rangle\rangle$, we have that
$\eta(\langle t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle, \langle \overline{a}, \langle v', \mathsf{L}\rangle\rangle) = (t_{v,v',a_1}^1, \langle t_{v,v',a_1}^1, \overline{q}, 1, v\rangle) \wedge$
$(t_{v,v',a_1}^1, \langle t_{v,v',a_1}^1, \overline{q}, 1, e\rangle)$.

(f) For every $\langle t_{v,v',a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle$ and $\langle \overline{a}, \langle v'', \mathsf{L}\rangle\rangle$, we have that
$\eta(\langle t_{v,v',a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle, \langle \overline{a}, \langle v'', \mathsf{L}\rangle\rangle) = (t_{v,v'',a_1}^1, \langle t_{v,v'',a_1}^1, \overline{q}, 1, v\rangle) \wedge$
$(t_{v,v'',a_1}^1, \langle t_{v,v'',a_1}^1, \overline{q}, 1, e\rangle)$.

(g) For every $\langle t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle$ and $\langle \overline{a}, \langle v', \mathsf{L}'\rangle\rangle$, we have that
$\eta(\langle t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle, \langle \overline{a}, \langle v', \mathsf{L}'\rangle\rangle) = (t_{v,v',a_1}^1, \langle t_{v,v',a_1}^1, \overline{q}, 1, e\rangle)$.

(h) For every $\langle t_{v,v',a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle$ and $\langle \overline{a}, \langle v'', \mathsf{L}'\rangle\rangle$, we have that
$\eta(\langle t_{v,v',a_1}^{|T_{v,a_1}|}, \overline{q}, 1, e\rangle, \langle \overline{a}, \langle v'', \mathsf{L}'\rangle\rangle) = (t_{v,v'',a_1}^1, \langle t_{v,v'',a_1}^1, \overline{q}, 1, e\rangle)$.

4. The transition function η is defined as follows:

(a) For every $\langle t_{v,v',a_1}^i, \overline{q}, 1, v\rangle$, $1 \le i \le |T_{v,v',a_1}|-1$ and a_j, $j \in \mathsf{L}$, we have that
$\eta(\langle t_{v,v',a_1}^i, \overline{q}, 1, v\rangle, \langle j, a_j\rangle) = (t_{v,v',a_1}^{i+1}, \langle \delta(v, t_{v,v',a_1}^i[j \leftarrow a_j]), t_{v,v',a_1}^{i+1}, \overline{q}, j, v\rangle) \wedge$
$(t_{v,v',a_1}^{i+1}, \langle t_{v,v',a_1}^{i+1}, \overline{q}, 1, v\rangle)$.

(b) For every $\langle t_{v,v',a_1}^{|T_{v,a_1}|}, \overline{q}, 1, v\rangle$, a_j, $j \in \mathsf{L}$ and $\langle v'', \mathsf{L}'\rangle$, we have that

i. $\eta(\langle t_{v,v',a_1}^{|T_{v,a_1}|}, \overline{q}, 1, v\rangle, \langle j, a_j\rangle) =$
$(\delta(v, t_{v,v',a_1}^i[j \leftarrow a_j]), \langle \delta(v, t_{v,v',a_1}^i[j \leftarrow a_j]), \overline{q}, j, v\rangle)$.

ii. $\eta(\langle t_{v,v',a_1}^{|T_{v,a_1}|}, \overline{q}, 1, v\rangle, \langle \langle j, a_j\rangle, \langle v'', \mathsf{L}'\rangle\rangle) =$
$(t_{v,v'',a_1}^1, \langle \delta(v, t_{v,v',a_1}^i[j \leftarrow a_j]), t_{v,v'',a_1}^1, \overline{q}, j, v\rangle)$.

(c) For every $\langle v, q_2, \ldots, q_k, j, v\rangle$, and $a_1 \in \kappa_1(v)$, we have that
$\eta(\langle v, q_2, \ldots, q_k, j, v\rangle, a_1) = \bigwedge_{\overline{q}\in\delta_2(q_2,\tau(v))\times\cdots\times\delta_k(q_k,\tau(v))}(t_{v,a_1}^1, \langle t_{v,a_1}^1, \overline{q}, j, v\rangle)$.

(d) For every $\langle t_{v,a_1}^i, \overline{q}, j, v\rangle$, $1 \le i \le |T_{v,a_1}|-1$ and $\langle a_2, \ldots, a_k\rangle$, we have that

$\eta(\langle t_{v,a_1}^i, \overline{q}, j, v\rangle, \langle a_2, \ldots, a_k\rangle) = (t_{v,a_1}^{i+1}, \langle \delta(t_{v,a_1}^i[j \leftarrow a_j]), t_{v,a_1}^{i+1}, \overline{q}, j, v\rangle) \wedge$
$(t_{v,a_1}^{i+1}, \langle t_{v,a_1}^{i+1}, \overline{q}, j, v\rangle)$.

(e) For every $\langle u, t_{v,a_1}^i, \overline{q}, j, v\rangle$, $1 \le i \le |T_{v,a_1}| - 1$ and $\langle a_2, \ldots, a_k\rangle$, we have that
$\eta(\langle u, t_{v,a_1}^i, \overline{q}, j, v\rangle, \langle a_2, \ldots, a_k\rangle) = (t_{v,a_1}^{i+1}, \langle u, t_{v,a_1}^{i+1}, \overline{q}, j, v\rangle)$.

(f) For every $\langle u, t_{v,v',a_1}^i, \overline{q}, j, v\rangle$, $1 \le i \le |T_{v,v',a_1}| - 1$ and $\langle a_2, \ldots, a_k\rangle$, we have that
$\eta(\langle u, t_{v,v',a_1}^i, \overline{q}, j, v\rangle, \langle a_2, \ldots, a_k\rangle) = (t_{v,v',a_1}^{i+1}, \langle u, t_{v,v',a_1}^{i+1}, \overline{q}, j, v\rangle)$.

(g) For every $\langle t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, j, v\rangle$, $\langle a_2, \ldots, a_k\rangle$ and $\langle v', \mathsf{L}'\rangle$ we have that:

i. $\eta(\langle t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, j, v\rangle, \langle a_2, \ldots, a_k\rangle) =$
$(\delta(v, t_{v,a_1}^{|T_{v,a_1}|}[j \leftarrow a_j]), \langle \delta(v, t_{v,a_1}^{|T_{v,a_1}|}[j \leftarrow a_j]), \overline{q}, j, v\rangle)$.

ii. $\eta(\langle t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, j, v\rangle, \langle \langle a_2, \ldots, a_k\rangle, \langle v', \mathsf{L}'\rangle\rangle) =$
$(t_{v,v',a_1}^1, \langle \delta(v, t_{v,a_1}^{|T_{v,a_1}|}[j \leftarrow a_j]), t_{v,v',a_1}^1, \overline{q}, j, v\rangle)$.

(h) For every $\langle u, t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, j, v\rangle$, $\langle a_2, \ldots, a_k\rangle$ and $\langle v', \mathsf{L}'\rangle$ we have that:

i. $\eta(\langle u, t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, j, v\rangle, \langle a_2, \ldots, a_k\rangle) = (u, \langle u, \overline{q}, j, v\rangle)$.

ii. $\eta(\langle u, t_{v,a_1}^{|T_{v,a_1}|}, \overline{q}, j, v\rangle, \langle\langle a_2, \ldots, a_k\rangle, \langle v', \mathsf{L}'\rangle\rangle) = (t_{v,v',a_1}^{1}, \langle u, t_{v,v',a_1}^{1}, \overline{q}, j, v\rangle)$.

5. $\alpha = \bigcup_{i \in \mathsf{L}} Q_{-i} \times \{\alpha_i\} \times \{i\} \times \{v\}$.