Perspective Games with Notifications

Orna Kupferman

School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel
orna@cs.huji.ac.il

Noam Shenwald

School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel
noam.shenwald@mail.huji.ac.il

Abstract

A reactive system has to satisfy its specification in all environments. Accordingly, design of correct reactive systems corresponds to the synthesis of winning strategies in games that model the interaction between the system and its environment. The game is played on a graph whose vertices are partitioned among the players. Starting from an initial vertex, the players jointly generate a computation, with each player deciding the successor vertex whenever the generated computation reaches a vertex she owns. The objective of the system player is to force the generated computation to satisfy a given specification. The traditional way of modelling uncertainty in such games is observation-based. There, uncertainty is longitudinal: the players partially observe all vertices in the history. Recently, researchers introduced *perspective games*, where uncertainty is transverse: players fully observe the vertices they own and have no information about the behavior of the computation between visits in such vertices. We introduce and study *perspective games with notifications*: uncertainty is still transverse, yet a player may be notified about events that happen between visits in vertices she owns. We distinguish between structural notifications, for example about visits in some vertices, and behavioral notifications, for example about the computation exhibiting a certain behavior. We study the theoretic properties of perspective games with notifications, and the problem of deciding whether a player has a winning perspective strategy. Such a strategy depends only on the visible history, which consists of both visits in vertices the player owns and notifications during visits in other vertices. We show that the problem is EXPTIME-complete for objectives given by a deterministic or universal parity automaton over an alphabet that labels the vertices of the game, and notifications given by a deterministic satellite, and is 2EXPTIME-complete for LTL objectives. In all cases, the complexity in the size of the graph and the satellite is polynomial – exponentially easier than games with observation-based partial visibility. We also analyze the complexity of the problem for richer types of satellites.

2012 ACM Subject Classification Formal languages and automata theory, Logic and verification

Keywords and phrases Games, Incomplete Information, Automata

Digital Object Identifier 10.4230/LIPIcs.FST & TCS 2020.2020.

1 Introduction

A reactive system has to satisfy its specification in all environments. Accordingly, design of correct reactive systems corresponds to the synthesis of a winning strategy for the system in a game that model the interaction between the system and its environment. The game is played on a graph whose vertices correspond to configurations along the interaction. We study here settings in which each configuration is controlled by either the system or its environment. Thus, the set of vertices is partitioned between the players, and the game is *turn-based*: starting from an initial vertex, the players jointly generate a *play*, namely a path in the graph, with each player deciding the successor vertex when the play reaches a vertex she controls. Each vertex is labeled by an assignment to a set AP of atomic propositions – these with respect to which the system is defined. The objective of the system is given by a language $L \subseteq (2^{AP})^\omega$, and it wins if the computation induced by the generated play, namely

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science.
Editors: Nitin Saxena and Sunil Simon; Article No. ; pp. :1–:25
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the word that labels its vertices, is in L [14, 4].

A *strategy* for a player directs her how to continue a play that reaches her vertices. We consider *deterministic* strategies, which choose a successor vertex. In games with *full visibility*, strategies may depend on the full history of the play. In games with *partial visibility*, strategies depend only on visible components of the history [16]. A well studied model of partial visibility is *observation based* [9, 6, 5, 2]. There, a player does not see the vertices of the game and can only observe the assignments to a subset of the atomic propositions. Accordingly, strategies cannot distinguish between different plays in which the observable atomic propositions behave in the same manner. Recently, [8] introduced *perspective games*. There, the visibility of each player is restricted to her vertices. Accordingly, a perspective strategy for a player cannot distinguish among histories that differ in visits to vertices owned by other players. As detailed in [8], the perspective model corresponds to switched systems and component-based software systems [1, 11, 12, 13].

Note that visibility and lack of visibility in the observation-based model are *longitudinal* – players observe all vertices, but partially. On the other hand, in the perspective model, players have full visibility on the parts of the system they control, and no visibility (in particular, even no information on the number of transitions taken) on the parts they do not control. Thus, visibility and lack of visibility are *transverse* – some vertices the players do not see at all, and some they fully see. For a comparison of perspective games with related visibility models (in particular, games with partial visibility in an asynchronous setting [15], switched systems [7], and control-flow composition in software and web service systems [12]), see [8].

In many settings, players indeed cannot observe the evolution of the computation in parts of the system they do not control, yet they may have information about events that happen during these parts. For example, if the system is synchronous with a global clock, then all players know the length of the invisible parts of the computation. Likewise, visits in some vertices of the other players may be observable, for example in a communication network in which all companies observe routers that belong to an authority and can detect visits to routers that leave a stamp. Finally, behaviors may be visible too, like an airplane that flies high, or a robot that enters a zone that causes an alarm to be activated. In this paper we introduce and study *perspective games with notifications*, which model such settings.

Formally, perspective games with notifications include, in addition to the game graph and the winning condition, an *information satellite*: a finite state machine that is executed in parallel with the game and may notify the players about events it monitors. We distinguish between *structural* satellites, which monitor the generated play, and *behavioral* satellites, which monitor the generated computation. Examples to structural satellites include ones that notify the players about visits in designated sets of states, transitions among regions in the system, say calls and returns in software systems, traversal of loops, etc. A typical behavioral satellite is associated with a regular language $R \subseteq (2^{AP})^*$. The satellite may notify the players whenever the computation induced by the play is in R (termed a *single-track* satellite), or whenever a suffix of the computation is in R (termed a *multi-track* satellite). The language R may vary from simple propositional assertion over AP, to rich finite on-going behaviors. Note that even very simple satellites may be very useful. For example, when $R = (2^{AP})^*$, the satellite acts as a clock, notifying the players about the length of the invisible parts of the computation.

We start by studying some theoretical aspects of perspective games with notifications. We consider two-player games with a winning condition $L \subseteq (2^{AP})^\omega$ such that Player 1 aims for a play whose computation is in L, and Player 2 aims for a play whose computation is not in L. Unsurprisingly, the basic features of the game are inherited from the model without notifications. In particular, perspective games with notifications are not determined. Thus, there are games in which Player 1 does not have a perspective strategy that forces the generated computation to satisfy L nor Player 2 has a perspective strategy that forces the generated computation not to satisfy L. Also, the restriction to a perspective strategy (as opposed to one that fully observes the computation) makes a difference only for one of the players. Thus, if Player 1 has a strategy to win against all perspective strategies of Player 2, she also has a perspective strategy to win against all strategies of Player 2.

The prime problem when reasoning about games is to decide whether a player has a winning strategy. Here the differences between perspective games and other models of partial visibility become significant: handling of observation-based partial visibility typically involves some subset-construction-like transformation of the game graph into a game graph of exponential size with full visibility. Accordingly, deciding of observation-based partial-visibility games is EXPTIME-complete in the graph [2, 6, 5, 3]. In perspective games, one can avoid this exponential blow-up in the size of the graph and trade it with an exponential blow-up in the (typically much smaller) winning condition [8].

Our main technical contribution is an extension of these good news to perspective games with notifications, and a study of the complexity in terms of the satellite. The solution in [8] is based on the definition of a tree automaton for winning strategies. The extension to a model with notifications is not easy, as the type of strategies is different. Let V_1 denote the set of vertices that Player 1 controls. With no notifications, a strategy for Player 1 is a function $f : V_1^* \to V$, mapping each visible history to a successor vertex. With notifications, the visible histories of Player 1 consist not only of vertices in V_1 but refer also to a set I of notifications that Player 1 may receive from the satellite. Moreover, histories that end in a notification in I correspond to vertices in the game in which Player 1 do not have control. Accordingly, the outcome of the strategy in them is not important, yet they should still be taken into account. We are still able to define a tree automaton for winning strategies. Essentially, the tree automaton follows both the satellite and the automaton for the winning condition, where a tree that encodes a strategy includes branches not only for vertices in V_1 but also branches for notifications in I. We analyze the complexity of our algorithm for winning conditions given by deterministic or universal co-Büchi or parity automata, as well as by LTL formulas, and show that the problem is EXPTIME-complete for all above types of automata and is 2EXPTIME-complete for LTL. In all cases, the complexity in terms of the graph and the satellite is polynomial.

While EXPTIME-hardness follows immediately from the setting with no notifications [8], we analyse the complexity also in terms of the satellite. Recall that given a finite language $R \subseteq (2^{AP})^*$, a satellite may be single-track, notifying about computations in R, or multi-track, notifying about computations in $(2^{AP})^* \cdot R$. We examine four cases, depending on whether the satellite is single- or multi-track and whether R is given by a deterministic or nondeterministic automaton. For deterministic single-track satellites, the complexity of deciding whether Player 1 wins is polynomial. In the other three cases, a naive construction of a satellite requires determinization and involves an exponential blow-up. Note that this applies also to the case where R is given by a deterministic automaton yet the satellite is multi-track, and thus has to follow all suffixes. We show that this blow up is unavoidable. Thus, deciding whether Player 1 wins is EXPTIME-hard even when the winning condition, which is the source for the exponential complexity in the setting with no notifications, is fixed. On the positive side, we show that many interesting cases need a fixed-size satellite, or a satellite whose state space can be merged with that of the game.

2 Preliminaries

2.1 Perspective games

A *game graph* is a tuple $G = \langle AP, V_1, V_2, v_0, E, \tau\rangle$, where AP is a finite set of atomic propositions, V_1 and V_2 are disjoint sets of vertices, owned by Player 1 and Player 2, respectively, and we let $V = V_1 \cup V_2$. Then, $v_0 \in V_1$ is an initial vertex, which we assume to be owned by Player 1, and $E \subseteq V \times V$ is a total edge relation, thus for every $v \in V$ there is $u \in V$ such that $\langle v, u\rangle \in E$. The function $\tau : V \to 2^{AP}$ maps each vertex to a set of atomic propositions that hold in it. The size $|G|$ of G is $|E|$, namely the number of edges in it.

In a beginning of a play in the game, a token is placed on v_0. Then, in each turn, the player that owns the vertex that hosts the token chooses a successor vertex and move there the token. A *play* $\rho = v_0, v_1, ...$ in G, is an infinite path in G that starts in v_0; thus for all $i \geq 0$ we have that $\langle v_i, v_{i+1}\rangle \in E$. The play ρ induces a *computation* $\tau(\rho) = \tau(v_0), \tau(v_1), ... \in (2^{AP})^\omega$.

A *game* is a pair $\mathcal{G} = \langle G, L\rangle$, where G is a game graph, and $L \subseteq (2^{AP})^\omega$ is a *behavioral winning condition*, namely an ω-regular language over the atomic propositions, given by an LTL formula or an automaton. Intuitively, Player 1 aims for a play whose computation is in L, while Player 2 aims for a play whose computation is in $comp(L) = (2^{AP})^\omega \setminus L$.

Let $\mathsf{Prefs}(G)$ be the set of nonempty prefixes of plays in G. For a sequence $\rho = v_0, \ldots, v_n$ of vertices, let $\mathsf{Last}(\rho) = v_n$. For $j \in \{1, 2\}$, let $\mathsf{Prefs}_j(G) = \{\rho \in \mathsf{Prefs}(G) : \mathsf{Last}(\rho) \in V_j\}$. In games with full visibility, the players have a full view of the generated play. Accordingly, a strategy for Player j maps $\mathsf{Prefs}_j(G)$ to vertices in V in a way that respects E. In *perspective games* [8], Player j can view only visits to V_j. Accordingly, strategies are defined as follows. For a prefix $\rho = v_0, \ldots, v_i \in \mathsf{Prefs}(G)$, and $j \in \{1, 2\}$, *the perspective of player j on ρ*, denoted $\mathsf{Persp}_j(\rho)$, is the restriction of ρ to vertices in V_j. We denote the perspectives of player j on prefixes in $\mathsf{Prefs}_j(G)$ by $\mathsf{PPrefs}_j(G)$, namely $\mathsf{PPrefs}_j(G) = \{\mathsf{Persp}_j(\rho) : \rho \in \mathsf{Prefs}_j(G)\}$. Note that $\mathsf{PPrefs}_j(G) \subseteq {V_j}^*$. A *perspective strategy* for player j, is then a function $f_j : \mathsf{PPrefs}_j(G) \to V$ such that for all $\rho \in \mathsf{PPrefs}_j(G)$, we have that $\langle \mathsf{Last}(\rho), f_j(\rho)\rangle \in E$. That is, a perspective strategy for player j maps her perspective of prefixes of plays that end in a vertex $v \in V_j$ to a successor of v.

The *outcome* of P-strategies f_1 and f_2 for Player 1 and Player 2, respectively, is the play obtained when the players follow their P-strategies. Formally, $\mathsf{Outcome}(f_1, f_2) = v_0, v_1, ...$ is such that for all $i \geq 0$ and $j \in \{1, 2\}$, if $v_i \in V_j$, then $v_{i+1} = f_j(\mathsf{Persp}_j(v_0, \ldots, v_i))$.

We use F and P to indicate the visibility type of strategies, namely whether they are full (F) or perspective (P). Consider a game $\mathcal{G} = \langle G, L\rangle$. For $\alpha, \beta \in \{F, P\}$, we say that Player 1 *$(\alpha, \beta)$-wins* $\mathcal{G}$ if there is an α-strategy f_1 for Player 1 such that for every β-strategy f_2 for Player 2, we have that $\tau(\mathsf{Outcome}(f_1, f_2)) \in L$. Similarly, Player 2 *$(\alpha, \beta)$-wins* $\mathcal{G}$ if there is an α-strategy f_2 for Player 2 such that for every β-strategy f_1 for Player 1, we have that $\tau(\mathsf{Outcome}(f_1, f_2)) \notin L$.

2.2 Automata

Given a set D of directions, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot c \in T$, where $x \in D^*$ and $c \in D$, 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 $x \cdot c$, for $c \in D$, are the *successors* of x. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in D$ such that $x \cdot c \in \pi$. 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.

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 α is an acceptance condition. We consider here the Büchi, co-Büchi, and parity acceptance conditions. 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) $\varepsilon \in T_r$ and $r(\varepsilon) = \langle \varepsilon, q_{in} \rangle$. (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 $0 \le i \le n-1$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = \langle x \cdot c_i, q_i \rangle$.

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. In Büchi and co-Büchi automata, the acceptance condition is $\alpha \subseteq Q$. A path π satisfies a Büchi condition α iff $inf(\pi) \cap \alpha \neq \emptyset$, and satisfies a co-Büchi condition α iff $inf(\pi) \cap \alpha = \emptyset$. In parity automata, the acceptance condition $\alpha : Q \to \{1, \ldots, k\}$ maps each vertex to a *color*. A path π satisfies a parity condition α iff the minimal color that is visited infinitely often in π is even. Formally, $\min\{i : inf(\pi) \cap \alpha^{-1}(i) \neq \emptyset\}$ 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 $\delta : Q \times \Sigma \to 2^Q$, and we often extend δ to sets of states and to finite words: for $S \subseteq Q$, we have that $\delta(S, \epsilon) = S$ and for a word $w \in \Sigma^*$ and a letter $\sigma \in \Sigma$, we have $\delta(S, w \cdot \sigma) = \delta(\delta(S, w), \sigma)$. When $\alpha \subseteq Q$, we are ometimes interested in reachability via a nonempty path that visits α. For this, we define $\delta_\alpha : 2^Q \times \Sigma^+ \to 2^Q$ as follows. First, $\delta_\alpha(S, \sigma) = \delta(S, \sigma) \cap \alpha$. Then, for a word $w \in \Sigma^+$, we define $\delta_\alpha(S, w \cdot \sigma) = \delta(\delta_\alpha(S, w), \sigma) \cup (\delta(S, w \cdot \sigma) \cap \alpha)$. Thus, either α is visited in the prefix of the run that reads w after leaving S, or the last state of the run is in α. It is not hard to prove by an induction on the length of w that for all states $q \in Q$, we have that $q \in \delta_\alpha(S, w)$ iff there is a run from S on w that reaches q and visits α after leaving S. We sometimes refer also to word automata on finite words. There, $\alpha \subseteq Q$ and a (finite) run is accepting if its last state is in α.

We denote each of the 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.

3 Perspective Games with Notifications

Consider a game graph $G = \langle AP, V_1, V_2, v_0, E, \tau \rangle$. An *information satellite* for G (*satellite*, for short) is finite-state machine $\mathcal{I} = \langle O, I, S, s_0, M, i_1, i_2 \rangle$, where O and I are *observation* and *information* alphabets, S is a finite set of states, $s_0 \in S$ is an initial state, $M : S \times O \to S$ is a deterministic transition function, and $i_1, i_2 : S \to I \cup \{\varepsilon\}$ are information functions for Players 1 and 2, respectively, where $\varepsilon \notin I$ is a special letter, standing for "no information". We distinguish between *structural* satellites, where $O = V$, and *behavioral* satellites, where $O = 2^{AP}$. Intuitively, the satellite is executed during the play, updating its state according to the current vertex or its label, possibly notifying the players with information in I.

▶ **Example 1.** Assume there is an atomic proposition $alarm \in AP$. Both players can hear whenever an alarm is activated, but they do not know for how many rounds it is on. A satellite that informs the players about the activation of the alarm is $\mathcal{I} = \langle 2^{\{alarm\}}, \{activated\}, S, s_0, M, i_1, i_2 \rangle$, with $S = \{s_0, s_1, s_2\}$, $M(s_i, \neg alarm) = s_0$, for all $i \in \{0, 1, 2\}$, $M(s_0, alarm) = s_1$, and $M(s_1, alarm) = M(s_2, alarm) = s_2$. Thus, the satellite moves to s_1 whenever a $\neg alarm \cdot alarm$ pattern is read, and then moves to and stays in s_2 as long as the alarm is on. When the alarm is deactivated, the satellite moves to s_0. Also, $i_1(s_1) = i_2(s_1) = activated$, and $i_1(s_0) = i_1(s_2) = i_2(s_0) = i_2(s_2) = \varepsilon$. Thus, when the satellite is in s_1, it notifies both players about the activation of the alarm. ◀

A *perspective game with notifications* is a tuple $\mathcal{G} = \langle G, \mathcal{I}, L \rangle$ where G and L are as in perspective games with no notifications, and $\mathcal{I} = \langle O, I, S, s_0, M, i_1, i_2 \rangle$ is a satellite. As in usual perspective games, Player 1 aims for a play whose computation is in L, while Player 2 aims for a play whose computation is in $comp(L)$. Now, however, the perspectives of the players contain, in addition to visits in their sets of vertices, also information from the satellite. Below we formalize this intuition.

We define the function $\zeta : V \to O$ that maps each vertex of G to the appropriate observation alphabet letter of $\mathcal{I}$. Thus, for every $v \in V$, we have that $\zeta(v) = v$ if $\mathcal{I}$ is structural, and $\zeta(v) = \tau(v)$ if $\mathcal{I}$ is behavioral. An *attributed path* in G is a sequence $\eta \in (V \times S)^*$ obtained by attributing a path $\rho = v_0, v_1, v_2, \ldots, v_n \in V^*$ in G by the state in S that $\mathcal{I}$ visits when a play proceeds along ρ. Formally, $\eta = \langle v_0, s_0 \rangle, \langle v_1, s_1 \rangle, \ldots, \langle v_n, s_n \rangle$ is such that for all $1 \leq i \leq n$, we have that $s_i = M(s_{i-1}, \zeta(v_i))$. Note that first the play proceeds from v_{i-1} to v_i, and then the satellite reads $\zeta(v_i)$ and proceeds accordingly. We use $\mathsf{Last}(\eta)$ to refer to v_n. Let $\mathsf{Prefs}^I(G) \subseteq (V \times S)^*$ be the set of nonempty attributed prefixes of plays in G. For $j \in \{1, 2\}$, let $\mathsf{Prefs}^I_j(G) = \{\eta \in \mathsf{Prefs}^I(G) : \mathsf{Last}(\eta) \in V_j\}$. For a prefix $\eta \in \mathsf{Prefs}^I(G)$, the *rich perspective of* Player j *on* η, denoted $\mathsf{Persp}^I_j(\eta)$, is the restriction of η to vertices in V_j and notifications of $\mathcal{I}$ that occur in vertices not in V_j. Formally, the function $info_j : (V \times S) \to V_j \cup I$ describes the information added to Player j in each round. For all $\langle v, s \rangle \in V \times S$, if $v \in V_j$, then $info_j(\langle v, s \rangle) = v$; if $v \notin V_j$, then $info_j(\langle v, s \rangle) = i_j(s)$. Note that in the latter case, it may be that $i_j(s) = \varepsilon$. Thus, if $\eta = \langle v_0, s_0 \rangle, \langle v_1, s_1 \rangle, \ldots, \langle v_n, s_n \rangle$, then $\mathsf{Persp}^I_j(\eta) = info_j(\langle v_0, s_0 \rangle) \cdot info_j(\langle v_1, s_1 \rangle) \cdots info_j(\langle v_n, s_n \rangle)$. Note that ε does not contribute letters to $\mathsf{Persp}^I_j(\eta)$, and so the length of $\mathsf{Persp}^I_j(\eta)$ is the number of the vertices in V_j in η plus the number of vertices not in V_j in which the satellite provides to Player j information in I.

▶ **Example 2.** Consider the alarm activation satellite described in Example 1, and consider a game graph G. Let $v_2^{\uparrow}$ and $v_2^{\downarrow}$ be vertices of Player 2 with $alarm \in \tau(v_2^{\uparrow})$ and $alarm \notin \tau(v_2^{\downarrow})$. Then, the rich perspective of Player 1 on the path $v_2^{\downarrow}, v_2^{\downarrow}, v_2^{\downarrow}, v_2^{\uparrow}, v_2^{\uparrow}, v_2^{\downarrow}, v_2^{\uparrow}, v_2^{\uparrow}, v_2^{\uparrow}, v_2^{\downarrow}$ is $\bullet, \bullet$, reflecting the two activations of the alarm during its traversal. Now, if $v_1^{\uparrow} \in V_1$, and $alarm \in \tau(v_1^{\uparrow})$, then the rich perspective of Player 1 on $v_2^{\downarrow}, v_2^{\downarrow}, v_1^{\uparrow}, v_2^{\downarrow}, v_2^{\uparrow}, v_2^{\uparrow}, v_1^{\uparrow}, v_2^{\downarrow}, v_1^{\uparrow}, v_2^{\uparrow}, v_2^{\uparrow}, v_2^{\uparrow}, v_2^{\downarrow}, v_1^{\uparrow}$ is $v_1^{\uparrow}, \bullet, v_1^{\uparrow}, v_1^{\uparrow}, v_1^{\uparrow}$. ◀

We denote the perspective of Player j on prefixes in $\mathsf{Prefs}_j^I(G)$ by $\mathsf{PPrefs}_j^I(G)$; thus $\mathsf{PPrefs}_j^I(G) = \{\mathsf{Persp}_j^I(\eta) : \eta \in \mathsf{Prefs}_j^I(G)\}$. A *perspective strategy* for Player j (P-strategy for short) is then a function $f_j : \mathsf{PPrefs}_j^I(G) \to V$ such that for all $\rho \in \mathsf{PPrefs}_j^I(G)$, we have that $\langle \mathsf{Last}(\eta), f_j(\eta) \rangle \in E$. That is, a perspective strategy for Player j maps her perspective prefixes of plays that end in a vertex $v \in V_j$ to a successor of v. The definitions of the outcome of F or P-strategies and F or P-winning are similar to the definitions in perspective games with no notifications, with Persp_j^I instead of Persp_j.

▶ **Example 3.** Consider the game graph G appearing in Figure 1. For simplicity, we assume that the atomic propositions in AP are mutually exclusive, and thus each vertex is labeled by a letter in $\Sigma = \{p, q, \#, \$\}$. Let $\mathcal{I}_1$ be a structural satellite that notifies Player 1 whenever a visit in w_q occurs, and let $\mathcal{I}_2$ be a behavioral satellite that notifies Player 1 whenever the computation induced so far ends in $\$ \cdot p$. Also, let φ_1 describe computations that every $q \cdot q$ subword is followed by a subword in $\Sigma \cdot q$, and every $q \cdot p$ is followed by $\Sigma \cdot p$. Formally, $\varphi_1 = G(((q \wedge Xq) \to XXXq) \wedge ((q \wedge Xp) \to XXXp))$. Likewise, let $\varphi_2 = G(((\$ \wedge Xp) \to XXXp) \wedge ((q \wedge Xp) \to XXXq))$.

As we elaborate in Appendix A.1, Player 1 cannot (P, F)-win $\langle G, \mathcal{I}_2, \varphi_1 \rangle$, yet she does (P, F)-win $\langle G, \mathcal{I}_1, \varphi_1 \rangle$. Also, Player 1 cannot (P, F)-win $\langle G, \mathcal{I}_1, \varphi_2 \rangle$, yet she does (P, F)-win $\langle G, \mathcal{I}_2, \varphi_2 \rangle$. ◀

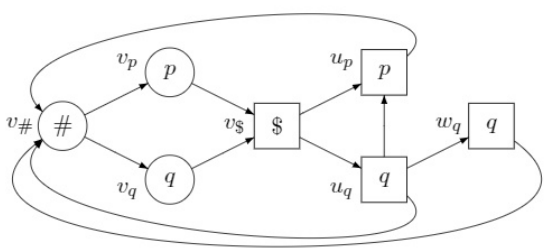

Figure 1 The game graph G over $AP = \{p, q, \#, \$\}$. The vertices of Player 1 are circles, and those of Player 2 are squares. The initial vertex is $v_\#$

Example 3 shows that, as is the case in perspective games with no notifications [8], P-strategies with no notifications are weaker than P-strategies with notifications, which are weaker than F-strategies. It also shows (see full proof in Appendix B.2) that perspective games with notifications are not determined. That is, there are perspective games with notifications where both Player 1 and Player 2 do not have P-winning strategies.

The following theorem states that the visibility type of Player 2 does not matter. Essentially (see Appendix B.1), it follows from the fact that if a perspective strategy of Player 1 loses against an F-strategy f_2 of Player 2, then it also loses to a P-strategy of Player 2 that is induced from f_2.

▶ **Theorem 4.** *For every perspective game with notifications $\mathcal{G}$, Player 1 (F, F)-wins $\mathcal{G}$ iff Player 1 (F, P)-wins $\mathcal{G}$, and Player 1 (P, F)-wins $\mathcal{G}$ iff Player 1 (P, P)-wins $\mathcal{G}$.*

Since the visibility type of Player 2 does not matter, we can omit it from our notation and talk about Player 1 P-winning a game. Also, specifying satellites, we remove the function i_2 from their description.

4 Deciding Perspective Games with Notifications

Consider a game $\mathcal{G} = \langle G, \mathcal{I}, L\rangle$, for a game graph $G = \langle AP, V_1, V_2, v_0, E, \tau\rangle$ and a satellite $\mathcal{I} = \langle O, I, S, s_0, M, i_1\rangle$. For a regular expression R over the alphabet V, an *R-path from v* is a finite path $v_1, \ldots, v_k \in L(R)$ in G such that $v_1 = v$. For a subset $X \subseteq V$, an *X^ω-path from v* is an infinite path $v_1, v_2, ... \in X^\omega$ in G with $v_1 = v$. Note, for example, that when Player 1 moves the token to a vertex $v \in V_2$, the token may traverse a $(V_2^+ \cdot V_1)$-path ρ from v, in which case it returns to V_1 in $\mathsf{Last}(\rho)$, or it my traverse a V_2^ω-path from v, in which case it never returns to a vertex in V_1. For a regular expression R over the alphabet $V \times S$, an *R-path from $\langle v, s\rangle$* is an attributed path $\langle v_1, s_1\rangle, \ldots, \langle v_k, s_k\rangle \in L(R)$ in G with $v_1 = v$ and $s_1 = s$. For such a path ρ, we denote its projections on V and S by $\rho|_V$ and $\rho|_S$, respectively.

Consider the satellite $\mathcal{I}$. For $\sigma \in I \cup \{\varepsilon\}$, we denote by S_σ the set of states in $\mathcal{I}$ in which Player 1 is notified σ. That is, $S_\sigma = \{s \in S : i_1(s) = \sigma\}$. Then, $S_I = \bigcup_{\sigma \in I} S_\sigma$ is the set of states in which Player 1 is notified some information. Equivalently, $S_I = S \setminus S_\varepsilon$.

We focus on games in which the winning condition L is given by a UCW. For simplicity, we denote them by $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U}\rangle$, for a UCW $\mathcal{U}$. Let $\mathcal{U} = \langle 2^{AP}, Q, q_0, \delta, \alpha\rangle$ In order for Player 1 to P-win $\mathcal{G}$, her objective in the beginning of the game is to force a token that is placed in v_0 into computations that $\mathcal{U}$ accepts from q_0 with the satellite being in state s_0. We can describe this objective by the triple $\langle v_0, q_0, s_0\rangle$. As the play progresses, the objective of Player 1 is updated. Moreover, as $\mathcal{U}$ is universal, the objective may contain several such triples. Below we formalize this intuition.

Consider a UCW $\mathcal{U} = \langle 2^{AP}, Q, q_0, \delta, \alpha\rangle$, a state $q \in Q$, and a state $s \in S$. Suppose that the token is placed in some vertex $v \in V_1$, the objective of Player 1 is to force the token into computations in $L(\mathcal{U}^q)$, and the satellite is in state s after seeing $\zeta(v)$. Assume further that Player 1 chooses to move the token to a successor v' of v and that $s' = M(s, \zeta(v'))$. We distinguish between two cases.

1. $v' \in V_1$. Then, the new objective of Player 1 is to force the token in v' into computations in $L(\mathcal{U}^{q'})$, for all states $q' \in \delta(q, \tau(v))$, with the satellite being in state s'.
2. $v' \in V_2$. Then, there are three cases:
   a. There is a ${V_2}^\omega$-path ρ from v' with $\tau(\rho) \notin L(\mathcal{U}^{q'})$ for some $q' \in \delta(q, \tau(v))$. We then say that v' *is a trap for* $\langle v, q\rangle$. Indeed, Player 2 can stay in vertices in V_2 and force the token into a computation not in $L(\mathcal{U}^{q'})$. Note that once Player 1 chooses a vertex that is a trap for $\langle v, q\rangle$, Player 2 has a strategy to win the game.
   b. v' is not a trap for $\langle v, q\rangle$, yet there is no $({V_2}^+ \cdot V_1)$-path from v'. That is, all paths from v' stay in vertices in V_2 and are in $L(\mathcal{U}^{q'})$ for all $q' \in \delta(q, \tau(v))$. We then say that v' *is safe for* $\langle v, q\rangle$. Indeed, Player 2 stays in vertices in V_2 and all the possible plays induce a computation in $L(\mathcal{U}^q)$. Note that once Player 1 chooses a safe vertex for $\langle v, q\rangle$, her objective is fulfilled regardless of the stragety of Player 2.
   c. v' is neither a trap nor safe for $\langle v, q\rangle$, in which case:
      i. For every $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v', s'\rangle$ Player 1 should force a token that is placed in v'' into computations in $L(\mathcal{U}^{q'})$, for all states $q' \in \delta(q, \tau(v \cdot \rho|_V))$, with the satellite being in state s''. Note that for all $\langle \hat{v}, \hat{s}\rangle$ along

ρ, we have $info_1(\langle \hat{v}, \hat{s}\rangle) = \varepsilon$, and so the visit in v'' is the first event that Player 1 observes after placing the token in v'.

ii. For every $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v', s'\rangle$, Player 1 should force a token that is placed in v'' with the satellite being in state s'' into computations in $L(\mathcal{U}^{q'})$, for all states $q' \in \delta(q, \tau(v \cdot \rho|_V))$. Note that for all $\langle \hat{v}, \hat{s}\rangle$ along ρ, we have $info_1(\langle \hat{v}, \hat{s}\rangle) = \varepsilon$, and so $i_1(s'')$ is the first event that Player 1 observes after placing the token in v'. Also note that ρ might be empty, in particular when Player 1 moves the token to a vertex in V_2 that invokes a notification of $\mathcal{I}$. In this case, $\langle v', s'\rangle = \langle v'', s''\rangle$.

The above analysis induces the definition of *updated objectives*: Consider a triple $\langle v, q, s\rangle \in V_1 \times Q \times S$, standing for an objective of Player 1 to force a token placed on v to be accepted by $\mathcal{U}^q$ with the satellite being in state s. For a successor v' of v, we define the set $S^{v'}_{v,q,s} \subseteq (V \times Q \times S \times \{\bot, \top\}) \cup \{\textbf{false}\}$ of objectives that Player 1 has to satisfy in order to fulfil her $\langle v, q, s\rangle$ objective after choosing to move the token to v'. Also, for a triple $\langle v, q, s\rangle \in V_2 \times Q \times S$, we define the set $S_{v,q,s} \subseteq V \times Q \times S \times \{\bot, \top\}$ of objectives that Player 1 has to satisfy in order to fulfil her $\langle v, q, s\rangle$ objective for every successor that Player 2 might choose for v. In both cases, the $\{\bot, \top\}$ flag in the objectives is used for tracking visits in α: an updated objective $\langle v'', q', s'', c\rangle \in S^{v'}_{v,q,s}$ has $c = \top$ if Player 2 can force a visit in α when $\mathcal{U}$ runs from q to q' along a word that labels a path from v via v' to v''.

Formally, for a triple $\langle v, q, s\rangle \in V \times Q \times S$ we define the set of updated objectives as follows. Let $s' = M(s, \zeta(v'))$.

1. If $v \in V_1$ and $E(v, v')$, we distinguish between three cases.
   a. If v' is a trap for $\langle v, q\rangle$, then $S^{v'}_{v,q,s} = \{\textbf{false}\}$.
   b. If v' is safe for $\langle v, q\rangle$, then $S^{v'}_{v,q,s} = \emptyset$.
   c. Otherwise, a tuple $\langle v'', q', s'', c\rangle$ is in $S^{v'}_{v,q,s}$ iff one of the following holds.
      i. $v' \in V_1$, $v'' = v'$, $q' \in \delta(q, \tau(v))$, and $s'' = s'$. Then, $c = \top$ iff $q' \in \alpha$.
      ii. $v' \in V_2$, and there is an $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v', s'\rangle$ such that $q' \in \delta(q, \tau(v \cdot \rho|_V))$. Then, $c = \top$ iff there is an $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v', s'\rangle$ such that $q' \in \delta_\alpha(q, \tau(v \cdot \rho|_V))$.
      iii. $v' \in V_2$, and there is an $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v', s'\rangle$ such that $q' \in \delta(q, \tau(v \cdot \rho|_V))$. Then, $c = \top$ iff there is an $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v', s'\rangle$ such that $q' \in \delta_\alpha(q, \tau(v \cdot \rho|_V))$.
2. If $v \in V_2$, a tuple $\langle v'', q', s'', c\rangle$ is in $S_{v,q,s}$ iff one of the following holds.
   a. There is an $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v, s\rangle$ such that $q' \in \delta(q, \tau(v \cdot \rho|_V))$. Then, $c = \top$ iff there is an $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v, s\rangle$ such that $q' \in \delta_\alpha(q, \tau(v \cdot \rho|_V))$.
   b. There is an $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v, s\rangle$ such that $q' \in \delta(q, \tau(v \cdot \rho|_V))$. Then, $c = \top$ iff there is an $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$-path $\rho \cdot \langle v'', s''\rangle$ from $\langle v, s\rangle$ such that $q' \in \delta_\alpha(q, \tau(v \cdot \rho|_V))$.

The notion of updated objectives is the key to our algorithm for deciding P-winning in perspective games with notifications. Recall that a perspective strategy for Player 1 is a function $f_1 : \mathsf{PPrefs}_1(G) \to V$ such that for all $\rho \in \mathsf{PPrefs}_1(G)$, we have that $\langle \mathsf{Last}(\rho), f_1(\rho)\rangle \in E$, where $\mathsf{PPrefs}_1(G)$ contains words in $V_1 \cup I$ that end with a vertex in V_1. Accordingly, we describe a strategy for Player 1 by a $(V \cup \{\oslash\})$-labeled $(V_1 \cup I)$-tree, where the letter $\oslash$ label nodes $x \notin \mathsf{PPrefs}_1(G)$, namely nodes $x \in (V_1 \cup I)^* \cdot I$. Formally, a $(V \cup \{\oslash\})$-labeled $(V_1 \cup I)$-tree $\langle (V_1 \cup I)^*, \eta\rangle$ is a P-strategy of Player 1 if for all $\rho \in (V_1 \cup I)^*$ and $v \in V_1$, we have that $\eta(\rho \cdot v) = v'$, where $v' \in V$ is such that $E(v, v')$, and for all $\sigma \in I$ we have that $\eta(\rho \cdot \sigma) = \oslash$, indicating Player 1 does not move the token when she receives the σ notification, and just keeps this notification in mind.

▶ **Theorem 5.** *Let $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U}\rangle$ be a game with notifications, where G is a game graph, $\mathcal{I} = \langle O, I, S, s_0, M, i_1\rangle$ is a satellite, and $\mathcal{U}$ is a UCW. We can construct a UCT $\mathcal{A}_\mathcal{G}$ over $(V \cup \{\oslash\})$-labeled $(V_1 \cup I)$-trees such that $\mathcal{A}_\mathcal{G}$ accepts a $(V \cup \{\oslash\})$-labeled $(V_1 \cup I)$-tree $\langle (V_1 \cup I)^*, \eta\rangle$ iff $\langle (V_1 \cup I)^*, \eta\rangle$ is a winning P-strategy for Player 1. The size of $\mathcal{A}_\mathcal{G}$ is polynomial in $|G|$, $|\mathcal{I}|$, and $|\mathcal{U}|$.*

Proof. Let $\mathcal{U} = \langle 2^{AP}, Q, q_0, \delta, \alpha\rangle$. We define $\mathcal{A}_\mathcal{G} = \langle V \cup \{\oslash\}, V_1 \cup I, Q', q'_0, \delta', \alpha'\rangle$, where:

1. $Q' = V \times Q \times S \times \{\perp, \top\}$. Intuitively, when $\mathcal{A}_\mathcal{G}$ is in state $\langle v, q, s, c\rangle$ it accepts strategies that force a token placed on v into a computation accepted by $\mathcal{U}^q$ with the satellite being in state s. The flag c is used for tracking visits in α.
2. $q'_0 = \langle v_0, q_0, s_0, \perp\rangle$.
3. The transitions are defined, for all states $\langle v, q, s, c\rangle \in V_1 \times Q \times S \times \{\perp, \top\}$, as follows.
   - **a.** If $v \in V_1$, then $\delta'(\langle v, q, s, c\rangle, \oslash) = \textbf{false}$, and for every $v' \in V$ we have the following transitions.
     - **i.** If $S^{v'}_{v,q,s} = \{\textbf{false}\}$ or $\neg E(v, v')$, then $\delta'(\langle v, q, s, c\rangle, v') = \textbf{false}$.
     - **ii.** If $S^{v'}_{v,q,s} = \emptyset$, then $\delta'(\langle v, q, s, c\rangle, v') = \textbf{true}$.
     - **iii.** Otherwise, $\delta'(\langle v, q, s, c\rangle, v') =$ $\bigwedge_{\{\langle v'', q', s'', c'\rangle \in S^{v'}_{v,q,s} : v'' \in V_1\}} (v'', \langle v'', q', s'', c'\rangle) \wedge \bigwedge_{\{\langle v'', q', s'', c'\rangle \in S^{v'}_{v,q,s} : v'' \in V_2\}} (i_1(s''), \langle v'', q', s'', c'\rangle)$.
   - **b.** If $v \in V_2$, then for all $v' \in V$, we have that $\delta'(\langle v, q, s, c\rangle, v') = \textbf{false}$. Also, $\delta'(\langle v, q, s, c\rangle, \oslash) =$ $\bigwedge_{\{\langle v'', q', s'', c'\rangle \in S_{v,q,s} : v'' \in V_1\}} (v'', \langle v'', q', s'', c'\rangle) \wedge \bigwedge_{\{\langle v'', q', s'', c'\rangle \in S_{v,q,s} : v'' \in V_2\}} (i_1(s''), \langle v'', q', s'', c'\rangle)$.

   Thus, for every updated objective $\langle v'', q', s'', c'\rangle$, the automaton $\mathcal{A}_\mathcal{G}$ sends a copy in state $\langle v'', q', s'', c'\rangle$ to direction v'' if $v'' \in V_1$, and to direction $i_1(s'')$, if $v'' \in V_2$. Note that several updated requirements may be sent to the same direction. In particular, in addition to multiple copies sent to the same direction due to universal branches in $\mathcal{U}$, a direction $\sigma \in I$ may "host" updated objectives associated with different vertices in V_2. Intuitively, such vertices are indistinguishable by Player 1.
4. $\alpha' = V \times Q \times S \times \{\top\}$. Recall that a $\top$ flag indicates that Player 2 may reach the Q-element in an updated objective traversing a path that visits α. Accordingly, the co-Büchi requirement to visit α only finitely many times amounts to a requirement to visit states with $\top$ only finitely many times.

◀

Theorem 5 gives us an upper bound on the problem of deciding whether Player 1 P-wins a perspective game with notifications.

▶ **Theorem 6.** *Deciding whether Player 1 P-wins a perspective game with notifications $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U}\rangle$, for a UCW $\mathcal{U}$, is EXPTIME-complete, and can be solved in time polynomial in $|G|$ and $|\mathcal{I}|$, and exponential in $|\mathcal{U}|$.*

Proof. Let $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U}\rangle$ and $\mathcal{I} = \langle O, I, S, s_0, M, i_1\rangle$. By Theorem 5, we can construct a UCT $\mathcal{A}_\mathcal{G}$ over $(V \cup \{\oslash\})$-labeled $(V_1 \cup I)$-trees such that $L(\mathcal{A}_\mathcal{G})$ is not empty iff there is a winning P-strategy for Player 1 in $\mathcal{G}$. The size of $\mathcal{A}_\mathcal{G}$ is polynomial in $|G|$, $|\mathcal{I}|$ and $|\mathcal{U}|$.

We construct an NBT $\mathcal{A}'_\mathcal{G}$ over $(V \cup \{\oslash\})$-labeled $(V_1 \cup I)$-trees such that $L(\mathcal{A}'_\mathcal{G})$ is not empty iff there is a winning P-strategy for Player 1 in $\mathcal{G}$. The size of $\mathcal{A}'_\mathcal{G}$ is polynomial in

$|G|$ and $|\mathcal{I}|$, and is exponential in $|\mathcal{U}|$. As we elaborate in Appendix B.3, the transformation from $\mathcal{A}_\mathcal{G}$ to $\mathcal{A}'_\mathcal{G}$ uses the fact that $\mathcal{A}_\mathcal{G}$ is deterministic in the V and S components, in order to generate, following the construction of [10], an NBT that it is polynomial in $|G|$ and $|\mathcal{I}|$ and exponential only in $|\mathcal{U}|$. Since the nonemptiness problem for an NBT can be solved in quadratic time, the specified complexity follows.

Since perspective games with notifications are a special case of perspective game (technically, with a satellite that only outputs ε), EXPTIME-hardness of the former implies an EXPTIME lower bound for our setting. ◀

Since an LTL ψ formula can be translated to a UCW $\mathcal{U}_\psi$ with an exponential blow up (for example, by translating $\neg\psi$ to an NBW [17], and then dualizing the NBW), Theorem 6 implies a 2EXPTIME upper bound for perspective games with notifications in which the winning condition is given by an LTL formula. Also, as has been the case in [8], it is possible to refine the $\{\bot, \top\}$ flag in the updated objectives to maintain the minimal parity color that is visited, and adjust the construction to games in which the winning condition is given by a UPW. The complexity stays exponential in the automaton. Formally, we have the following.

▶ **Theorem 7.** *Deciding whether* Player 1 *P-wins a perspective game with notifications* $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U}\rangle$*, for a UPW* $\mathcal{U}$*, is EXPTIME-complete, and can be solved in time polynomial in* $|G|$ *and* $|\mathcal{I}|$*, and exponential in* $|\mathcal{U}|$*.*

Proof. The updated objectives defined for the case where the winning condition is given by a UCW contain a flag that records visits in the co-Büchi condition. When $\mathcal{U}$ is a UPW with k colors, we define the flag such that it records the minimal color visited instead. That is, $S^{v'}_{v,q,s}, S_{v,q,s} \subseteq (V \times Q \times S \times \{1, ..., k\}) \cup \{\textbf{false}\}$, is such that for every updated objective $\langle v'', q', s'', c\rangle \in S^{v'}_{v,q,s} \cup S_{v,q,s}$, Player 2 can force a path from v (via v') to v'' in which the minimal color visited in the run of $\mathcal{U}$ along it from q to q' is c. We then use a construction that is similar to the one in the proof of Theorem 5 to construct a UPT $\mathcal{A}_\mathcal{G}$ over $(V \cup \{\oslash\})$-labeled $(V_1 \cup I)$-trees such that $L(\mathcal{A}_\mathcal{G})$ is not empty iff there is a winning P-strategy for Player 1 in $\mathcal{G}$. The size of $\mathcal{A}_\mathcal{G}$ is polynomial in $|G|$, $|\mathcal{I}|$ and $|\mathcal{U}|$.

By [10], APT emptiness can be reduced to UCT emptiness with a polynomial blow up. From there, determinizm in the V-component implies the required complexity. ◀

5 Examples of Information Satellites

Consider a game graph $G = \langle AP, V_1, V_2, v_0, E, \tau\rangle$. Recall that a *structural satellite* for G is a satellite $\mathcal{I} = \langle O, I, S, s_0, M, i_1\rangle$ with $O = V$. Thus, the satellite can view the state in which the play is, and can decide about outputs to Player 1 based on this visibility. Then, a *behavioral satellite* for G has $O = 2^{AP}$. Thus, the satellite can only observe the labels of vertices, and its outputs to Player 1 are based only on these labels. In this section we describe some natural structural and behavioral satellites.

5.1 Structural Information Satellites

A visible subset of vertices As discussed in Section 1, in some settings there is a subset of vertices $I_1 \subseteq V_2$ such that Player 1 is notified whenever the play visits a vertex in I_1. Then, the satellite is $\langle V, I_1, V, v_0, M, i_1\rangle$, where for all $v, u \in V$, we have that $M(v, u) = u$, $i_1(v) = v$ if $v \in I_1$, and $i_1(v) = \varepsilon$, otherwise. Thus, the state of the satellite follows the vertex of the game, and it produces an output during visits in I_1. Note that Player 1 is notified not only about visits in I_1, but also about the specific vertex that is visited. Alternatively, we could define the satellite with output *in* only, $i_1(v) = in$ if $v \in I_1$, and $i_1(v) = \varepsilon$, otherwise. Here, Player 1 is notified that some vertex in I_1 has been visited, with no information about which vertex it is.

Observation-based uncertainty Assume that there is a subset of the atomic propositions $AP_1 \subseteq AP$, such that Player 1 observes the assignments to AP_1 in Player 2's vertices. A corresponding satellite is $\langle V, 2^{AP_1}, V, v_0, M, i_1\rangle$, where for all $v, u \in V$, we have that $M(v, u) = u$, $i_1(v) = \tau(v) \cap AP_1$ if $v \in V_2$, and $i_1(v) = \varepsilon$, otherwise. Note that this case combines the transverse visibility of perspective games with the longitudinal visibility in observation-based games. Indeed, when the token is in Player 2's vertices, Player 1's visibility is information based. In particular, Player 1 does know the number of states visited. It is not hard to see that when $AP_1 = AP$, then, as the winning condition is behavioral, the setting coincides with games with full visibility. Also, note that even though the notifications of the satellite are in 2^{AP_1}, we could not define it as a behavioral information satellite.

Visible switches among regions Assume that the vertices in V_2 is partitioned into disjoint *regions* $V_2^1, \ldots, V_2^k$. For example, the regions may correspond to modules or procedures. In Appendix A.2, we describe satellites that notify Player 1 upon entry to the different regions. Here too, the satellite may declare the exact region or just notify about a switch. In the appendix we also describe an interesting variant of the above – a satellite that notifies Player 1 whenever Player 2 loops in a vertex.

5.2 Behavioral Information Satellites

Visible regular properties Assume there is a property, given by a regular language R over 2^{AP}, such that Player 1 is notified whenever the computation generated since the beginning of the play is in R. For example, if $AP = \{p, q\}$, the property may be $\mathbf{true}^* \cdot p \cdot (\neg q)^*$, thus we want to notify Player 1 whenever a vertex satisfying p has been visited with no visit in a vertex satisfying q following this visit. Then, if $A_R = \langle 2^{AP}, S, s_0, M, F\rangle$ is a DFW that recognizes R, an appropriate satellite is $\mathcal{I} = \langle 2^{AP}, \{\bullet\}, S, M(s_0\tau(v_0)), M, i_1\rangle$, where for every $s \in S$, we have that $i_1(s) = \bullet$ if $s \in F$, and $i_1(s) = \varepsilon$, otherwise. Note that the initial state of the satellite is the state of A_R after reading the label of v_0. Indeed, notifications inform Player 1 about the membership of the computation up to (and including) the vertex where the token visits. A useful special case of regular properties are these of the form $\mathbf{true}^* \cdot R$, for a regular language R over 2^{AP}. Thus, Player 1 is notified whenever the computation generated since the beginning of the play has a suffix in R. As we discuss in Section 6, handling of the two types of notifications is of different complexity.

As we detail in Appendix A.3, the above can be generalized to multiple regular languages $R_1, \ldots, R_k$ over 2^{AP}, where for every $1 \le i \le k$, Player 1 is notified whenever the computation generated since the beginning of the play is in R_i.

Then, if for every $1 \le i \le k$, the DFW $A_i = \langle 2^{AP}, S_i, s_i^0, M_i, F_i\rangle$ recognize R_i, then an appropriate satellite is $\mathcal{I} = \langle 2^{AP}, 2^{\{\bullet_1,\ldots,\bullet_k\}}, S, s^0, M, i_1\rangle$ is such that $S = S_1 \times S_2 \times \cdots \times S_k$, $s^0 = \langle M_1(s_1^0, \tau(v_0)), \ldots, M_k(s_k^0, \tau(v_0))\rangle$, the transitions are as in a usual product of automata, and for every $\langle s_1, s_2, \ldots, s_k\rangle \in S$ and $1 \le i \le k$, we have that $\bullet_i \in i_1(\langle s_1, s_2, \ldots, s_k\rangle)$ iff $s_i \in F_i$.

A clock A step-counter notifies Player 1 how many vertices of Player 2 are visited between visits in her own vertices. This is done by a behavioral satellite for the regular language $R = (2^{AP})^*$. Indeed, then, Player 1 is notified in every step.

6 Complexity for the Different Satellites

Recall that the complexity of deciding a game depends on the size of the satellite. Formally, for a satellite $\mathcal{I} = \langle O, I, S, s_0, M, i_1, i_2\rangle$, the state space of the NBT whose nonemptiness we check in Theorem 6 is a product of S with other parameters. In this section we study the size of different satellites, and the way it affects the complexity.

We start with structural satellites. It is easy to see that the structural satellites described in Section 5.1 are such that $S = V$ or $S = V \times C$, for some constant set C. Moreover, since the satellite follows the play (formally, in all states of the UCT constructed in Theorem 5, the V-component agrees with the V-component of S. Accordingly, we do need the V-component in the state space and can maintain C only. In other words, the state space of $\mathcal{A}_\mathcal{G}$ can be redefined as $V \times Q \times C \times \{\perp, \top\}$, and the complexity of the decision problem is reduced accordingly.

We continue to simple behavioral satellites. One is the clock from Section 5.2, which involves a satellite with a single state, leading to $\mathcal{A}_\mathcal{G}$ with state space $V \times Q \times \{\perp, \top\}$, and a simpler definition of updated objectives. Another easy special case are *propositional satellites*, which notify Player 1 whenever the play visits a vertex v such that $\tau(v) \models \theta$, for an assertion θ over AP. Indeed, for such notifications we need a two-state satellite. We note that in both cases, EXPTIME-hardness of the game is valid. While the case of propositional satellites this follows by an easy reduction from the case of perspective games with no notifications, for the case of clocks such a reduction is impossible. Nevertheless, the reduction in the lower bound proof in [8] suits are needs, since the game constructed in there alternates between V_1 and V_2. Such a game has full visibility, and thus it also has a clock inherinlty.

Our focus in this section is general behavioral satellites. Consider a regular language R. We distinguish between the case where the satellite notifies Player 1 whenever the computation since the beginning of the game is in R (termed *single-track* satellites, as they follow a single computation), and the case where the satellite notifies Player 1 whenever a suffix of the computation is in R, or equivalently, whenever the computation is in $\mathbf{true}^* \cdot R$ (termed *multi-track* satellites, as they follow all suffixes of the computation). Analyzing the complexity of games with behavioral satellites, we assume a game is given by a tuple $\mathcal{G} = \langle G, A_R, \mathcal{U}, t\rangle$, where G and $\mathcal{U}$ are the game graph and winning condition, A_R is the *pattern automata*, namley the automata describing a regular property R, and $t \in \{\textsc{single}, \textsc{multi}\}$, is a flag indicating whether the satellite is single- or multi-track.

▶ **Theorem 8.** *Deciding whether* Player 1 *P-wins in a game $\mathcal{G} = \langle G, \mathcal{A}_R, \mathcal{U}, t\rangle$ can be solved in time polynomial in $|G|$, exponential in $|\mathcal{U}|$, and*
- *polynomial in $|\mathcal{A}_R|$ when $t = \textsc{single}$ and A_R is a DFW.*
- *exponential in $|\mathcal{A}_R|$ when $t = \textsc{multi}$ or A_R is an NFW. Moreover, the problem is EXPTIME-complete already for a fixed-size $\mathcal{U}$.*

Proof. The upper bounds follow from Theorem 6, and the fact we can generate from A_R a satellite with no blow-up when $t = \textsc{single}$ and A_R is a DFW, and a satellite exponential in A_R when $t = \textsc{multi}$ or A_R is an NFW. Note that when $t = \textsc{multi}$, we first add to A_R a $\mathbf{true}^*$ self-loop leading to the initial state, which makes it nondeterministic.

We continue to the EXPTIME lower bound, and start with the case $t = \textsc{single}$ and A_R is an NFW. We describe a reduction from linear-space alternating Turing machines (ATM). The details of the reduction can be found in Appendix B.4.2. Given an ATM M and a word $w \in \Gamma^*$, we construct a game $\mathcal{G} = \langle G, A_R, \mathcal{U}, \textsc{single}\rangle$ such that Player 1 P-wins $\mathcal{G}$ iff M accepts w. The size of $\mathcal{U}$ is fixed, and G and A_R are of size linear in $s(n)$ where $n = |w|$. Essentially, Player 1 generates a legal accepting computation in the computation tree of M on w. Thus Player 1 chooses successors in existential configurations, and Player 2 chooses successors in universal ones. The challenging part of the reduction is to guarantee that the sequence of configurations generated is a legal computation, and to do it with a fixed size winning condition. We encode a configuration of M by a string $\#\gamma_1\gamma_2\cdots(q,\gamma_i)\cdots\gamma_{s(n)}$. When $\mathcal{U}$ is polynomial, it is easy to relate letters in the same address in successive configurations, making sure that the transition function of M is respected. When $\mathcal{U}$ is of a fixed size, it is not clear how to do it, as such letters are $s(n)$-letters apart. The key idea is to use A_R in order to do the required counting: We let Player 2 choose an address $k \in \{1,\ldots,s(n)\}$ and challenge Player 1 by raising a flag whenever the address is k. The winning condition $\mathcal{U}$ checks that the transition function of M is respected whenever the flag is raised, which forces Player 1 to respect the transitions function of M in address k. Moreover, since Player 1 does not know k, she has to always respect the transition function. The above mechanism is not sufficient, as Player 2 may try to fail Player 1 by raising the flag maliciously, that is, not sticking to one address k. This is where the notifications enter the picture: the language R detects malicious flag raises and notifies Player 1 about them. For this, A_R has to count to $s(n)$, but this is allowed, and enables $\mathcal{U}$ to skip the counting. In addition, $\mathcal{U}$ restricts the check of Player 1 only to ones in which the flag is raised properly.

Then, when $t = \textsc{multi}$ and A_R is a DFW (or NFW), the reduction is similar and is based on the fact that the only nondeterminism in A_R above is in guessing malicious flag raises, namely raises that are not $s(n)$ letters apart. Such a behavior can be specified by a regular expression $\mathbf{true}^* \cdot R$ for R that can be described by a DFW of size polynomial in $s(n)$. ◀

References

1. S. Agarwal, M. S. Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *Proc. 32nd IEEE International Conference on Computer Communications*, pages 2211–2219, 2013.
2. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.
3. D. Berwanger, K. Chatterjee, M. De Wulf, L. Doyen, and T. A. Henzinger. Strategy construction for parity games with imperfect information. *Information and Computation*, 208(10):1206–1220, 2010.
4. R. Bloem, K. Chatterjee, and B. Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking.*, pages 921–962. Springer, 2018.
5. K. Chatterjee and L. Doyen. The complexity of partial-observation parity games. In *Proc. 16th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning*, pages 1–14. Springer, 2010.
6. K. Chatterjee, L. Doyen, T. A. Henzinger, and J-F. Raskin. Algorithms for ω-regular games with imperfect information. In *Proc. 15th Annual Conf. of the European Association for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 287–302, 2006.
7. D. Fisman and O. Kupferman. Reasoning about finite-state switched systems. In *5th International Haifa Verification Conference*, volume 6405 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2009.
8. O. Kupferman and G. Vardi. Perspective games. In *Proc. 34th IEEE Symp. on Logic in Computer Science*, pages 1 – 13, 2019.
9. O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.
10. O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

11 D. Liberzon. *Switching in Systems and Control*. Birkhauser, 2003.
12 Y. Lustig and M.Y. Vardi. Synthesis from component libraries. *Software Tools for Technology Transfer*, 15(5-6):603–618, 2013.
13 M. Margaliot. Stability analysis of switched systems using variational principles: an introduction. *Automatica*, 42(12):2059–2077, 2006.
14 D.A. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.
15 B. Puchala. Asynchronous omega-regular games with partial information. In *35th Int. Symp. on Mathematical Foundations of Computer Science*, pages 592–603. Springer, 2010.
16 J.H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and Systems Science*, 29:274–301, 1984.
17 M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

A Examples

A.1 A detailed version of Example 3

Consider the game graph G appearing in Figure 1. Note that whenever the token reaches $v_\$$, there are four possible sub-computations it may generate before returning to $v_\#$; these are $\$ \cdot p \cdot \#$, $\$ \cdot q \cdot \#$, $\$ \cdot q \cdot p \cdot \#$ and $\$ \cdot q \cdot q \cdot \#$. Let $\mathcal{G}_1 = \langle G, \varphi_1 \rangle$ be a perspective game with $\varphi_1 = G(((q \wedge Xq) \rightarrow XXXq) \wedge ((q \wedge Xp) \rightarrow XXXp))$. It is easy to see that Player 1 cannot (P, F)-win $\mathcal{G}_1$, because she is unable to distinguish between the different possible sub-computations, and thus every P-strategy of hers chooses the same successor of $v_\#$ for all four cases. Now consider the perspective game with notifications $\mathcal{G}'_1 = \langle G, \mathcal{I}_1, \varphi_1 \rangle$ where $\mathcal{I}_1$ is a structural satellite that notifies Player 1 whenever a visit in w_q occurs. The information from the satellite restricts the possibilities; when Player 1 gets a notification, she knows that the last sub-computation is $\$ \cdot q \cdot q \cdot \#$. When she does not get a notification, she knows that the last sub-computation could be any option from the rest of them. Obviously, Player 1 (P, F)-wins $\mathcal{G}'_1$, because Player 1 can distinguish between the sub-computations $\$ \cdot q \cdot q \cdot \#$ and $\$ \cdot q \cdot p \cdot \#$, and she can choose the successor of $v_\#$ after every visit in it accordingly.

Let $\mathcal{G}_2 = \langle G, \varphi_2 \rangle$ be a perspective game with $\varphi_2 = G(((\$ \wedge Xp) \rightarrow XXXp) \wedge ((q \wedge Xp) \rightarrow XXXq))$. Player 1 cannot (P, F)-win $\mathcal{G}_2$, for the same reason she cannot (P, F)-win $\mathcal{G}_1$. Now consider the perspective game with notifications $\mathcal{G}'_2 = \langle G, \mathcal{I}_2, \varphi_2 \rangle$ where $\mathcal{I}_2$ is a behavioral satellite that notifies Player 1 whenever the computation induced so far is a word in the regular language $(p + q + \# + \$)^* \cdot \$ \cdot p$. Now, when Player 1 get a notification, it indicates that the last sub-computation is $\$ \cdot p \cdot \#$, and when she doesn't get a notification, she knows that the last sub-computation could be any option from the rest of them. Obviously, Player 1 (P, F)-wins $\mathcal{G}'_2$, because Player 1 can distinguish between the sub-computations $\$ \cdot p \cdot \#$ and $\$ \cdot q \cdot p \cdot \#$, and she can choose the successor of $v_\#$ after every visit in it accordingly.

Note that Player 1 cannot P-win the games $\langle G, \mathcal{I}_1, \varphi_2 \rangle$ and $\langle G, \mathcal{I}_2, \varphi_1 \rangle$, since $\mathcal{I}_1$ adds the same information for both $\$ \cdot p \cdot \#$ and $\$ \cdot q \cdot p \cdot \#$ sub-computations, and $\mathcal{I}_2$ adds the same information for both $\$ \cdot q \cdot q \cdot \#$ and $\$ \cdot q \cdot p \cdot \#$ sub-computations, so in both games any P-strategy of Player 1 chooses the same successor of $v_\#$ for both cases.

A.2 Structural satellites for visible switches among regions

Assume that the vertices in V_2 is partitioned into disjoint *regions* $V_2^1, \ldots, V_2^k$. For example, the regions may correspond to modules or procedures. If Player 1 is notified upon entry to

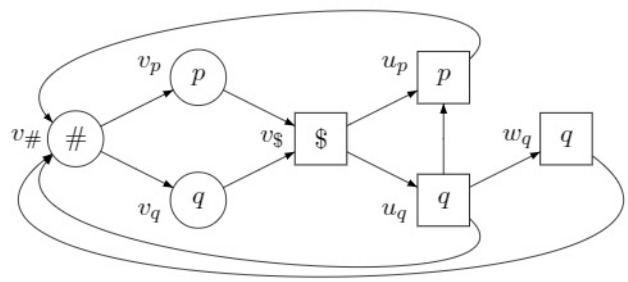

Figure 2 The game graph G over $AP = \{p, q, \#, \$\}$. The vertices of Player 1 are circles, and those of Player 2 are squares. The initial vertex is $v_\#$

the different regions, then the corresponding satellite is $\langle V, \{1, \ldots, k\}, S, \langle v_0, \circ\rangle, M, i_1\rangle$, where $S = (V_1 \times \{\circ\}) \cup (V_2 \times \{\circ, \bullet\})$. Thus, the state space of the satellite has one copy of the vertices in V_1 and two copies of the vertices in Player 2. Then, M and i_1 are as follows. For a vertex $v \in V_2$, let $reg(v)$ be the region of v; thus $v \in V_2^{reg(v)}$. Then, for all $v, u \in V$ and $j \in \{\circ, \bullet\}$, we have that $M(\langle v, j\rangle, u) = \langle u, \circ\rangle$ if $u \in V_1$ or $reg(v) = reg(u)$, and $M(\langle v, j\rangle, u) = \langle u, \bullet\rangle$ if $reg(v) \neq reg(u)$. Also, for every $\langle v, j\rangle \in S$ we have that $i_1(\langle v, j\rangle) = reg(v)$ if $j = \bullet$, and $i_1(\langle v, j\rangle) = \varepsilon$, otherwise. As in the case of a visible subset of vertices, the satellite can notify Player 1 only about a switch in a region, without specifying which region it is. Then, the satellite has only output $\bullet$, and $i_1(\langle v, j\rangle) = \bullet$ if $j = \bullet$, and $i_1(\langle v, j\rangle) = \varepsilon$, otherwise. Note that in both case, Player 1 is not notified about the number of rounds that Player 2 is spending in each region, and only about switches among them.

An interesting variant of the above is a satellite that notifies Player 1 whenever Player 2 loops in a vertex. Note that this is a special case of the above, where each vertex of V_2 has its own region, with a dual $\{\circ, \bullet\}$ notification. Namely, we let Player 1 know when there is no change in the region. Then, the satellite is $\langle V, \{\bullet\}, S, \langle v_0, \circ\rangle, M, i_1\rangle$, where i_1 is as above, yet for every $v, u \in V$ and $j \in \{\circ, \bullet\}$, we have that $M(\langle v, j\rangle, u) = \langle u, \circ\rangle$ if $u \in V_1$ or $v \neq u$, and $M(\langle v, j\rangle, u) = \langle u, \bullet\rangle$, otherwise.

A.3 Behavioral satellites for multiple regular languages

Let $R_1, \ldots, R_k$ be regular languages over 2^{AP}, where for every $1 \leq i \leq k$, we want Player 1 to be notified whenever the computation generated since the beginning of the play is in R_i.

Then, if for every $1 \leq i \leq k$, the DFW $A_i = \langle 2^{AP}, S_i, s_i^0, M_i, F_i\rangle$ recognize R_i, then an appropriate satellite is $\mathcal{I} = \langle 2^{AP}, 2^{\{\bullet_1, \ldots, \bullet_k\}}, S, s^0, M, i_1\rangle$ is such that $S = S_1 \times S_2 \times \cdots \times S_k$, $s^0 = \langle M_1(s_1^0, \tau(v_0)), \ldots, M_k(s_k^0, \tau(v_0))\rangle$, the transitions are as in a usual product of automata, and for every $\langle s_1, s_2, \ldots, s_k\rangle \in S$ and $1 \leq i \leq k$, we have that $\bullet_i \in i_1(\langle s_1, s_2, \ldots, s_k\rangle)$ iff $s_i \in F_i$.

B Proofs

B.1 Proof of Theorem 4

Let $\mathcal{G} = \langle G, \mathcal{I}, L\rangle$. First, consider an F or P strategy f_1 of Player 1, and assume that $\tau(\mathsf{Outcome}(f_1, f_2)) \in L$ for every F-strategy f_2 of Player 2. Clearly, $\tau(\mathsf{Outcome}(f_1, f_2)) \in L$ for every P-strategy f_2 of Player 2.

For the other direction, consider an F or P strategy f_1 of Player 1, and assume that $\tau(\mathsf{Outcome}(f_1, f_2)) \notin L$ for some F-strategy f_2 of Player 2. Let $\rho = \mathsf{Outcome}(f_1, f_2)$. We define an P-strategy f_2' for Player 2 such that for every prefix ρ' of ρ with $\mathsf{Last}(\rho') \in V_2$ we have $f_2'(\mathsf{Persp}_2^I(\rho')) = f_2(\rho')$. Note that for every two distinct prefixes ρ', ρ'' of ρ with $\mathsf{Last}(\rho'), \mathsf{Last}(\rho'') \in V_2$, the lengths of $\mathsf{Persp}_2^I(\rho')$ and $\mathsf{Persp}_2^I(\rho'')$ are different, thus f_2' is well defined. Now, as $\mathsf{Outcome}(f_1, f_2') = \mathsf{Outcome}(f_1, f_2)$, we have that $\tau(\mathsf{Outcome}(f_1, f_2')) \notin L$, and we are done.

B.2 Perspective games with notifications are not determined

Consider the perspective game with notifications $\langle G, \mathcal{I}_1, \varphi_2\rangle$ described in Example 3. As argued above, Player 1 does not P-win the game. In addition, as Player 1 does F-win $\langle G, \mathcal{I}_1, \varphi_2\rangle$, we have that Player 2 does not P-win $\langle G, \mathcal{I}_1, \neg\varphi_2\rangle$.

B.3 The transition to an NBT in the proof of Theorem 6

For $k \geq 1$, let $[k] = \{1, ..., k\}$. The construction in [10] transforms the UCT $\mathcal{A}_{\mathcal{G}} = \langle V \cup \{\oslash\}, V_1 \cup I, Q', q_0', \delta', \alpha'\rangle$ to an NBT with states $W = 2^{Q' \times [k]} \times 2^{Q' \times [k]}$, where k is such that $|Q'| \cdot k$ bounds the size an NRT that is equivalent to $\mathcal{A}_{\mathcal{G}}$, which is exponential in $|Q'|$. Also, for every state $\langle P, O\rangle \in W$, we have that $O \subseteq P$, and if $\langle q, i\rangle$ and $\langle q', i'\rangle$ are in P with $q = q'$, then $i = i'$. Therefore, the states in W can be written as $2^{Q'} \times 2^{Q'} \times \mathcal{F}$, where $\mathcal{F}$ is the set of functions $f : Q' \to [k]$. Recall that the states of the UCT $\mathcal{A}_{\mathcal{G}}$ are $Q' = V \times Q \times S \times \{\bot, \top\}$, and that $\mathcal{A}_{\mathcal{G}}$ is deterministic in the V and S components. Hence, the translation of $\mathcal{A}_{\mathcal{G}}$ to an NRT is polynomial in $|G|$ and $|\mathcal{I}|$, and exponential in $|U|$, and thus k is only polynomial in $|G|$ and $|\mathcal{I}|$. Also, for every $\langle P, O\rangle \in W$, if $\langle v, q, s, c, i\rangle$ and $\langle v', q', s', c', i'\rangle$ are in P, then since $\mathcal{A}_{\mathcal{G}}$ is deterministic in the V and S component, we have that $v = v'$ and $s = s'$. Therefore, the states in W can be written as $V \times S \times 2^{Q \times \{\bot,\top\}} \times 2^{Q \times \{\bot,\top\}} \times \mathcal{F}$, where $\mathcal{F}$ is the set of functions $f : Q \times \{\bot, \top\} \to [k]$. Hence, $|W|$ is polynomial in $|G|$ and $|\mathcal{I}|$, and exponential in $|\mathcal{U}|$.

B.4 Lower Bounds

The reductions in Sections B.4.1 and B.4.2 are from the membership problem for linear-space alternating Turing machines (ATM), defined below.

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 at 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 $\#(q_{init}, w_1) \cdot ... \cdot w_n \cdot \sqcup^{s(n)-n}$, 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 successors of a configuration $\#\gamma_1\gamma_2...(q, \gamma_i), \ldots, \gamma_{s(n)}$ 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 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, in both proofs we assume that M behaves this way.

B.4.1 Lower bound for a clock

We show a reduction from the membership problem for a linear-space alternating Turing machine (ATM). Given an ATM $M = \langle Q_e, Q_u, \Gamma, \Delta, q_{init}, q_{acc}, q_{rej} \rangle$ and a word $w \in \Gamma^*$, we construct a game with a clock $\mathcal{G} = \langle G, \mathcal{U} \rangle$ such that M accepts w iff Player 1 has a winning P-strategy in $\mathcal{G}$. We first describe the game graph G; The vertices of Player 1 are going to maintain information about the last transition (in particular, the current state of M), but no information about the tape content. The vertices of Player 2 are going to maintain information about the last transition and the letter under the reading head. In each Player 1 turn, she chooses a transition in Δ that corresponds to the current state and letter, and moves to a Player 2 vertex accordingly. Since the current letter is not encoded in Player 1's vertices, then Player 1 might lie, but then the DFW would make sure that she looses the game. Also, the Player 2 vertex that Player 1 chooses to move to must correspond to the current letter. Again, if Player 1 lies about it, then the DFW makes sure she looses the game. In a Player 2 turn, she chooses a transition according to the current state and letter - both encoded in her vertices, and moves to a corresponding Player 1 vertex. Recall that the transitions in M alternate between existential and universal states. Accordingly, there is exactly one Player 2 vertex between two Player 1 vertices in the play. This fact enables Player 1 to maintain the tape configuration although she sees only her vertices, and makes $\mathcal{G}$ a game with full visibility, and thus it is also has a clock.

We continue to the winning condition $\mathcal{U}$. Intuitively, $\mathcal{U}$ makes sure that Player 1 does not lie about the current letter, both when choosing her transitions, and when passing the control to Player 2. Since there are exponentially many possible tape content. Instead, $\mathcal{U}$ maintains only the letter in some specific position $0 \leq k \leq s(|w|) - 1$ on the tape. The position k is chosen by Player 2 during a preamble we add to the game. Player 1 does not see the preamble, and thus she does not know k. Accordingly, in order to avoid loosing, Player 1 should not lie about any of the tape cells and thus should faithfully simulate the computation of M on w. Hence, Player 1 has a winning P-strategy iff M accepts w.

B.4.2 Proof of the lower bounds in Theorem 8

We describe a reduction from linear-space alternating Turing machines (ATM). Given an ATM M and a word w with $n = |w|$, we construct a game $\mathcal{G} = \langle G, A_R, \mathcal{U}, \textsc{single}\rangle$, such that $\mathcal{U}$ is fixed-size, G and A_R are of size linear in $s(n)$, and Player 1 P-wins $\mathcal{G}$ iff M accepts w. We first explain the main ideas of the reduction, and then describe the formal definitions of G, A_R, and $\mathcal{U}$. Note that the winning condition $\mathcal{U}$ is on finite words. Also, it is an NFW and the upper bound is for UFW or DFW, but since it is of a fixed size, also a deterministic version of it is of a fixed size.

Essentially, Player 1 generates a legal accepting computation in the computation tree of M on w. Thus Player 1 chooses successors in existential configurations, and Player 2 chooses successors in universal ones. The challenging part of the reduction is to guarantee that the sequence of configurations generated is a legal computation, and to do it with a fixed size winning condition. When $\mathcal{U}$ is polynomial, it is easy to relate letters in the same address in successive configurations, making sure that the transition function of M is respected. When $\mathcal{U}$ is of a fixed size, it is not clear how to do it, as such letters are $s(n)$-letters apart. The key idea is to use A_R in order to do the required counting: We let Player 2 choose an address $k \in \{1, \ldots, s(n)\}$ and challenge Player 1 by raising a flag whenever the address is k. The winning condition $\mathcal{U}$ checks that the transition function of M is respected whenever the flag is raised, which forces Player 1 to respect the transitions function of M in address k. Moreover, since Player 1 does not know k, she has to always respect the transition function. The above mechanism is not sufficient, as Player 2 may try to fail Player 1 by raising the flag maliciously, that is, not sticking to one address k. This is where the notifications enter the picture: the language R detects malicious flag raises and notifies Player 1 about them. For this, A_R has to count to $s(n)$, but this is allowed, and enables $\mathcal{U}$ to skip the counting. In addition, $\mathcal{U}$ restricts the check of Player 1 only to ones in which the flag is raised properly.

Assuming the players form a valid branch of a valid computation tree, then if M accepts w, the branch reaches an accepting configuration. Also, if M rejects w then Player 2 is able to choose successors of universal configurations that lead to a rejecting configuration. That way, if the objective of Player 1 is to reach an accepting configuration, she P-wins $\mathcal{G}$ iff M accepts w.

The challenge here is to force Player 1 to construct a correct branch in a computation tree of M on w with a winning condition of 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 $\#\sigma_1 \ldots \sigma_{s(n)} \# \sigma'_1 \ldots \sigma'_{s(n)}$ 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. $next_l(\langle \gamma_{i-1}, \gamma_i, \gamma_{i+1}\rangle) = next_l(\langle \#, \gamma_i, \gamma_{i+1}\rangle) = next_l(\langle \gamma_{i-1}, \gamma_i, \#\rangle) = \gamma_i$.
2. $next_l(\langle (q, \gamma_{i-1}), \gamma_i, \gamma_{i+1}\rangle) = next_l(\langle (q, \gamma_{i-1}), \gamma_i, \#\rangle) = \begin{cases} \gamma_i & \text{If } ((q, \gamma_{i-1}), \langle (q', \gamma'_{i-1}, L), (q_r, \gamma_r, d_r)\rangle) \in \Delta \\ (q', \gamma_i) & \text{If } ((q, \gamma_{i-1}), \langle (q', \gamma'_{i-1}, R), (q_r, \gamma_r, d_r)\rangle) \in \Delta \end{cases}$
3. $next_l(\langle \gamma_{i-1}, (q, \gamma_i), \gamma_{i+1}\rangle) = next_l(\langle \#, (q, \gamma_i), \gamma_{i+1}\rangle) = next_l(\langle \gamma_{i-1}, (q, \gamma_i), \#\rangle) = \gamma'_i$ where $((q, \gamma_i), \langle (q', \gamma'_i, d), (q_r, \gamma_r, d_r)\rangle) \in \Delta$.
4. $next_l(\langle \gamma_{i-1}, \gamma_i, (q, \gamma_{i+1})\rangle) = next_l(\langle (\#, \gamma_i, (q, \gamma_{i+1})\rangle) = \begin{cases} \gamma_i & \text{If } ((q, \gamma_{i-1}), \langle (q', \gamma'_{i+1}, R), (q_r, \gamma_r, d_r)\rangle) \in \Delta \\ (q', \gamma_i) & \text{If } ((q, \gamma_{i-1}), \langle (q', \gamma'_{i+1}, L), (q_r, \gamma_r, d_r)\rangle) \in \Delta \end{cases}$
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. Checking the consistency with $next_l$ and $next_r$ for every position in the computation cannot be achieved by a fixed size NFW, so the size limit of the winning condition makes it impossible to force Player 1 to form the valid computation. This is because it must compare between the same address along the entire computation, for all the addresses of the working tape, which induce space complexity polynomial in $s(n)$. We work around it by using a secret checkup. Player 2 can choose an address $1 \leq k \leq s(n)$ without Player 1 knowing, and let the winning condition check the consistency of the k-th cell between consecutive configurations by raising a flag whenever the address is k. In order to keep Player 2 from raising the flag maliciously and letting the winning condition compare between different addresses in consecutive configurations, the pattern automata monitors her behavior so Player 1 could reverse choices that leads to that. Since the wanted behavior of Player 2 is cyclic, were the length of the cycle is $s(n)$, we can construct such pattern automata NFW of size polynomial in $s(n)$.

First we describe the game graph. During the game, the players are forming a branch of a computation tree of M on w; Player 2 chooses an annotation for the current letter of the configuration indicating whether the flag is raised and the winning condition should test the consistency of the current address between consecutive configurations or not, by choosing "1" or "0", respectively. Once Player 2 marks an address k by "1", we say that she is *fair* if from now on she marks the k-th tape cell by "1" and the other tape cells by "0"; otherwise, we say she is *unfair*. After Player 2 chooses an annotation, Player 1 has the option to reverse the choice of Player 2 by using the negation character " $\sim$ " or to keep it by using the character "$\checkmark$", without knowing what was her choice or what is the outcome of reversing it. Then, Player 1 chooses the letter of the current address, and the process repeats. 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. The same way, Player 2 chooses the direction of the successor configuration after every universal configuration. Thus, the play induce a sequence that is alternating between 1/0 annotations, tape cell content and branching choices, that form a sequence of consecutive configurations of M that are a branch of a computation tree of M on w: $...0\#\{d_1\}0\gamma_1 0\gamma_2 ...0(q,\gamma_i)...1\gamma_k...0\gamma_{s(n)}0\#\{d_2\}0\gamma'_1 0\gamma'_2...0(q',\gamma'_j)...1\gamma'_k...0\gamma'_{s(n)}...$, where $d_1, d_2 \in \{l, r\}$. At the entrance to the game, Player 1 is forced "hard-coded" to form the initial configuration of M on w, while the annotation mechanism is enabled. Note that this is the part of the game that causes the polynomial complexity, and the necessity of that will be explained shortly.

Next we describe the pattern automata NFW A_R. Intuitively, We want to know when Player 2 is being unfair and tries to fail Player 1 by raising the flag maliciously, causing the winning condition to compare two different addresses in consecutive configurations, in order for Player 1 to be able to reverse such choices. So, A_R accepts every word that is not a prefix of a word in the language $L(0^* \cdot (1 \cdot 0^{s(n)})^*)$. This is a simplified description where the letters of the tape content and the branching choices are omitted. Moreover, if Player 1 chooses to reverse Player 2 annotation upon A_R's notification, the modified annotation is considered fair. Namely, the sequence $0\cdot \sim$ is equal to the annotation $1 \cdot \checkmark$ and $1\cdot \sim$ is equal to $0 \cdot \checkmark$. Note that if Player 1 is forming a correct branch of a computation tree, she can always reverse unfair annotation of Player 2 and so nothing prevents her from winning the game, assuming M does accept w, of course. Such NFW of size linear in $s(n)$ can be easily constructed.

Finally we describe the winning condition NFW $\mathcal{U}$. Intuitively, we want to force Player 1 to form a correct branch of a computation tree of M on w, and for that purpose we want the annotations to force consistency with $next_l$ and $next_r$; Assuming Player 2 is fair, she raises the flag whenever the address is k by marking every k tape cell by 1 for some $0 \leq k \leq s(n)$ starting from some configuration, and all the other tape cell by 0. Since k is not known to PLAYER 1 and neither is the configuration that the checkup is starting from, if $\mathcal{U}$ forces consistency with $next_l$ or $next_r$ between any two consecutive 1 annotations, she must form the a correct branch of a computation tree with respect to the branching choices, otherwise she might lose. There are four conditions that PLAYER 1 has to fulfil in order to P-win the game:

1. The computation should start from the initial configuration.
2. The computation should be consistent with $next_l$ between consecutive flag raises with the l branching choice between them.
3. The computation should be consistent with $next_r$ between consecutive flag raises with the r branching choice between them.
4. The computation should reach an accepting configuration.

Note that a winning condition of fixed size cannot force an unconstrained computation to start from the initial configuration while supporting the checkup mechanism, since that requires separate attention to every possible choice of PLAYER 2 of an address in the initial configuration to start the secret checkup. This is the reason we use the game itself to force the computation to start from the initial configuration.

When M accepts w, it is in PLAYER 1's best interest to form the correct configurations with respect to the branching choices and reverse unfair annotations of PLAYER 2. When M does not accept w, PLAYER 1 cannot win, even if she is arriving at a vertex that corresponds with q_{acc}. This is simply because she does not know the position of the secret checkup, and reversing fair annotations might not help; When PLAYER 1 reverses a fair annotation, she doesn't know if it was an 0 annotation or 1 annotation, and that can lead to forcing consistency with *next* between two different address unknown to PLAYER 1. If PLAYER 1 tries to lie about the content, and the first address that she is trying to choose the incorrect letter is k, then PLAYER 2 can choose this k to be the address to raise the flag upon.

Now we specify the formal definitions of G, A_R and $\mathcal{U}$.

1. The game graph $G = \langle AP, V_1, V_2, v_0, E, \tau \rangle$ is defined as follows:
   a. $AP = \Sigma \cup \{\$, \sim, \checkmark, 1, 0, l, r\}$. the APs are mutually exclusive, so we view them as the alphabet instead of 2^{AP}.
   b. $V_1 = \{v_0\} \bigcup_{t \in \{e,u\}} \{v_t, v_t^{\sim}, v_t^{\checkmark}, l_t, r_t\} \cup \bigcup_{0 \leq i \leq s(n)} \{v_i^{\$}, v_i^{\sim}, v_i^{\checkmark}, w_i\} \cup \bigcup_{\sigma \in (Q_e \times \Gamma) \cup \Gamma \cup \{\#\}} \{v_e^{\sigma}\} \cup \bigcup_{\sigma \in (Q_u \times \Gamma) \cup \Gamma} \{v_u^{\sigma}\}$.
   The vertex v_0 is the initial vertex. The vertices $\bigcup_{\sigma \in (Q_e \times \Gamma) \cup \Gamma \cup \{\#\}} \{v_e^{\sigma}\}$ are *the existential content vertices*, that are used to form the existential configurations, and have the *informer vertex* v_e, the *reverse vertex* $v_e^{\sim}$, and the *preserve vertex* $v_e^{\checkmark}$ as their own annotation-reversing mechanism. The same way, $\bigcup_{\sigma \in (Q_u \times \Gamma) \cup \Gamma} \{v_u^{\sigma}\}$, v_u, $v_u^{\sim}$, and $v_u^{\checkmark}$ are *the universal content vertices* and their annotation-reversing mechanism.
   Upon arriving to an informer vertex, PLAYER 1 finds out whether PLAYER 2 chose the fair annotation. After that PLAYER 1 chooses either to reverse the annotation by moving to the appropriate reverse vertex or to keep the annotation by moving to the preserve vertex, and then she chooses a letter.
   The vertices $\bigcup_{0 \leq i \leq s(n)} \{v_i^{\$}, v_i^{\sim}, v_i^{\checkmark}, w_i\}$ are the vertices that form the initial configuration. For every $0 \leq i \leq s(n)$, the vertex w_i represent the i-th letter in the initial configuration, and the vertices $v_i^{\$}, v_i^{\sim}$ and $v_i^{\checkmark}$ are its separate annotation-reversing mechanism.
   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 from by moving to l_e or r_e

from $v_e^{\#}$, and at the end of an universal configuration, Player 2 makes that choice at the vertex $v_u^{\#}$, by choosing either l_u or r_u. From both l_e and r_e, Player 1 moves to ν_u, to start the successor universal configuration. The same way, from both l_u and r_u, Player 1 moves to ν_e, to start the successor existential configuration.

c. $V_2 = \bigcup_{t\in\{e,u\}}\{\nu_t, \nu_t^0, \nu_t^1\} \cup \bigcup_{0\le i\le s(n)}\{\nu_i, \nu_i^0, \nu_i^1\} \cup \{v_u^{\#}\}$. The vertices $\{\nu_e, \nu_e^0, \nu_e^1\}$ are the annotation mechanism of the existential configurations, where at ν_e Player 2 chooses the annotation for the current letter by either moving to ν_e^1 or ν_e^0, which annotate the letter as the supervised letter or an unsupervised letter, respectively. The vertices $\{\nu_u, \nu_u^0, \nu_u^1\}$ are the annotation mechanism of the universal configurations, and the vertices $\{\nu_i, \nu_i^0, \nu_i^1\}$ are the annotation mechanism of the i-th letter in the initial configuration. Finally, the vertex $v_u^{\#}$ is the vertex that represent the end of an universal configuration, and upon arriving to it, Player 2 chooses what direction to proceed from by moving to l_u or r_u.

d. The set E contains the following edges:

i. $\langle v_0, \nu_0\rangle$.

ii. For every $0 \le i \le s(n)$ we have the following edges:

- $\langle \nu_i, \nu_i^0\rangle$ and $\langle \nu_i, \nu_i^1\rangle$.
- $\langle \nu_i^0, v_i^{\$}\rangle$ and $\langle \nu_i^1, v_i^{\$}\rangle$.
- $\langle v_i^{\$}, v_i^{\sim}\rangle$ and $\langle v_i^{\$}, v_i^{\checkmark}\rangle$.
- $\langle v_i^{\sim}, w_i\rangle$ and $\langle v_i^{\checkmark}, w_i\rangle$.

iii. $\langle w_i, \nu_{i+1}\rangle$ for every $0 \le i \le s(n) - 1$.

iv. $\langle w_{s(n)}, \nu_e\rangle$.

v. For every $t \in \{e, u\}$ we have the following edges:

- $\langle \nu_t, \nu_t^0\rangle$ and $\langle \nu_t, \nu_t^1\rangle$.
- $\langle \nu_t^0, v_t\rangle$ and $\langle \nu_t^1, v_t\rangle$
- $\langle v_t, v_t^{\sim}\rangle$ and $\langle v_t, v_t^{\checkmark}\rangle$.
- $\langle v_t^{\sim}, v_t^{\sigma}\rangle$ and $\langle v_t^{\checkmark}, v_t^{\sigma}\rangle$ for every $\sigma \in (Q_t \times \Gamma) \cup \Gamma \cup \{\#\}$.
- $\langle v_t^{\sigma}, \nu_t\rangle$ for every $\sigma \in (Q_t \times \Gamma) \cup \Gamma$.
- $\langle v_t^{\#}, l_t\rangle$ and $\langle v_t^{\#}, r_t\rangle$.
- $\langle l_t, \nu_{t'}\rangle$ and $\langle r_t, \nu_{t'}\rangle$ where $t' = \{e, u\} \setminus \{t\}$.

e. The labeling of the vertices is as follows:

i. $\tau(v) = \$$ for every $v \in \{v_0\} \cup \bigcup_{0\le i\le s(n)}\{\nu_i, v_i^{\$}\} \cup \bigcup_{t\in\{e,u\}}\{\nu_t, v_t\}$.

ii. $\tau(v) = \sim$ for every $v \in \bigcup_{0\le i\le s(n)}\{v_i^{\sim}\} \cup \bigcup_{t\in\{e,u\}}\{v_t^{\sim}\}$.

iii. $\tau(v) = \checkmark$ for every $v \in \bigcup_{0\le i\le s(n)}\{v_i^{\checkmark}\} \cup \bigcup_{t\in\{e,u\}}\{v_t^{\checkmark}\}$.

iv. $\tau(v) = 0$ for every $v \in \bigcup_{0\le i\le s(n)}\{\nu_i^0\} \cup \bigcup_{t\in\{e,u\}}\{\nu_t^0\}$.

v. $\tau(v) = 1$ for every $v \in \bigcup_{0\le i\le s(n)}\{\nu_i^1\} \cup \bigcup_{t\in\{e,u\}}\{\nu_t^1\}$.

vi. $\tau(v_t^{\sigma}) = \sigma$ for every $\sigma \in (Q_t \times \Gamma) \cup \Gamma\{\#\}$ and $t \in \{e, u\}$.

vii. $\tau(v) = l$ for every $v \in \{l_e, l_u\}$.

viii. $\tau(v) = r$ for every $v \in \{r_e, r_u\}$.

ix. $\tau(w_i) = w_i$ where w_i is the i-th letter in the initial configuration and $0 \le i \le s(n)$.

2. The NFW pattern automata $A_R = \langle AP, S, s_{init}, M, S_{acc}\rangle$ is defined as follows:

a. The states set S contain the following states:

i. s_{init} and $s^1_{\mathbf{false}}$. the state $s^1_{\mathbf{false}}$ is used to identify 1 annotations that are reversed before the first unchanged 1 annotation.

ii. s^1 indicating reading the first unchanged 1 annotation.

iii. s_i^0 for every $1 \le i \le n$ indicating how many 0 annotations were read after the last 1 annotation.

iv. $S_{acc} = \{s_i^{acc} : 0 \leq i \leq s(n)\} \cup \{s_{acc}\}$ indicating reading an unfair annotation after the i-th annotation starting from the latest 1 annotation.

b. The transition function M defined as follows:

i. $M(s, \sigma) = s$ for every $s \in S$ and $\sigma \in \Sigma \cup \{\$, \checkmark, l, r\}$. $M(s_{acc}, \sigma) = s_{acc}$ for every $\sigma \in AP$.

ii. $M(s_{init}, 0) = s_{init}$.

iii. $M(s_{init}, 1) = \{s^1, s^1_{\textbf{false}}\}$.

iv. $M(s^1_{\textbf{false}}, \sim) = s_{init}$.

v. $M(s_{init}, \sim) = s^1$.

vi. $M(s^1, 0) = s^0_1, M(s^0_{s(n)}, 1) = s^1$, and $M(s^0_i, 0) = s^0_{i+1}$ for every $1 \leq i \leq s(n) - 1$.

vii. $M(s^1, 1) = s^{acc}_0, M(s^0_{s(n)}, 0) = s^{acc}_{s(n)}$ and $M(s^0_i, 1) = s^{acc}_i$ for every $1 \leq i \leq s(n) - 1$.

viii. $M(s^{acc}_i, \sim) = s^0_{i+1}$ for every $0 \leq i \leq s(n) - 1$, and $M(s^{acc}_{s(n)}, \sim) = s^1$.

ix. $M(s, \sigma) = s_{acc}$ for every $s \in S_{acc} \setminus \{s_{acc}\}$ and $\sigma \in AP \setminus \{\sim\}$.

3. The NFW winning condition $\mathcal{U} = \langle AP, W, w_{init}, \delta, W_{acc}\rangle$ is defined as follows:

a. First, we define $\delta(w, \$) = w$ for every $w \in W$.

b. Next, we attend to the requirement of consistency between consecutive 1 annotations with respect to the branching choice. For every $(\langle \sigma_1, \sigma_2, \sigma_3\rangle, d) \in (\Sigma \times (\Sigma \setminus \{\#\}) \times \Sigma) \times \{l, r\}$ we define a subset of W called $W^{\sigma_1,\sigma_2,\sigma_3,d}$:

i. The states of the component are: b^x, $b^x_{\textbf{false}}$, $b^x_{\textbf{true}}$, ${\sigma_1}^x$, 1^x, $1^x_{\textbf{false}}$, $1^x_{\textbf{true}}$, ${\sigma_2}^x$, 0^x, $0^x_{\textbf{false}}$, $0^x_{\textbf{true}}$, σ_3, e^x, $e^x_{\textbf{false}}$ and $e^x_{\textbf{true}}$, where $x = (\sigma_1, \sigma_2, \sigma_3, d)$.
Upon entering the component, we stay at the *beginning* state b^x, waiting for the beginning of the sequence $0\sigma_1 1\sigma_2 0\sigma_3$. The component guesses when the sequence begins, and then move to ${\sigma_1}^x$ indicating we expect σ_1, from there to 1^x to read 1, to ${\sigma_2}^x$, 0^x,${\sigma_3}^x$ to read the sequence $\sigma_2 0 \sigma_3$, and then move to the *exit* state of the component e^x. We then stay at e^x until the end of the current configuration.
The states $b^x_{\textbf{false}}$, $b^x_{\textbf{true}}$, $1^x_{\textbf{false}}$, $1^x_{\textbf{true}}$,$0^x_{\textbf{false}}$, $0^x_{\textbf{true}}$, $e^x_{\textbf{false}}$ and $e^x_{\textbf{true}}$, are for dealing with the annotation-reversing mechanism. For example, assume that when we read 0 in state s, we move to state s'. Recall that both sequences $0 \cdot \checkmark$ and $1 \cdot \sim$ are considered the same. Then, upon reading 0, we move to state $s_{\textbf{true}}$ and then expect to read $\checkmark$ in order to proceed to s'. In a symmetrical manner, upon reading 1 we move to $s_{\textbf{false}}$, and then expect to read $\sim$ in order to proceed to s'.

ii. The definition of the transitions between those states describes the behavior specified earlier:

- $\delta(b^x, \sigma) = b^x$ for every $\sigma \in \Sigma \setminus \{\#\}$.
- $\delta(b^x, 0) = b^x_{\textbf{true}}$ and $\delta(b^x, 1) = b^x_{\textbf{false}}$.
- $\delta(b^x_{\textbf{true}}, \checkmark) = \delta(b^x_{\textbf{false}}, \sim) = \{b^x, {\sigma_1}^x\}$.
- $\delta({\sigma_1}^x, \sigma_1) = 1^x$.
- $\delta(1^x, 0) = 1^x_{\textbf{false}}$ and $\delta(1^x, 1) = 1^x_{\textbf{true}}$.
- $\delta(1^x_{\textbf{true}}, \checkmark) = \delta(1^x_{\textbf{false}}, \sim) = {\sigma_2}^x$.
- $\delta({\sigma_2}^x, \sigma_2) = 0^x$.
- $\delta(0^x, 0) = 0^x_{\textbf{true}}$ and $\delta(0^x, 1) = 0^x_{\textbf{false}}$.
- $\delta(0^x_{\textbf{true}}, \checkmark) = \delta(0^x_{\textbf{false}}, \sim) = {\sigma_3}^x$.
- $\delta({\sigma_3}^x, \sigma_3) = e^x$.
- $\delta(e^x, \sigma) = e^x$ for every $\sigma \in \Sigma \setminus \{\#\}$.
- $\delta(e^x, 0) = e^x_{\textbf{true}}$ and $\delta(e^x, 1) = e^x_{\textbf{false}}$.
- $\delta(e^x_{\textbf{true}}, \checkmark) = \delta(e^x_{\textbf{false}}, \sim) = e^x$.

The d parameter indicate that we are currently reading the configuration $succ_d(c)$ where c is the previous configuration, and thus $\delta(w, d) = w$ for all $w \in W^{\sigma_1,\sigma_2,\sigma_3,d}$, which implies that reading $\{l, r\} \setminus \{d\}$ causes the computation to be rejected.

Then, for every $x = (\sigma_1, \sigma_2, \sigma_3, d) \in \Sigma^3 \times \{l, r\}$, we have that $\delta(e^x, \{\varepsilon, \#\}) = \{b^y : y \in \Sigma \times \{next_{d'}(\langle\sigma_1, \sigma_2, \sigma_3\rangle)\} \times \Sigma \times \{d'\}, d' \in \{l, r\}\}$. Namely, after the end of the current configuration, we can continue from $W^{\sigma_1,\sigma_2,\sigma_3,d}$ to any other component that is expecting to see $next_l(\langle\sigma_1, \sigma_2, \sigma_3\rangle)$ or $next_r(\langle\sigma_1, \sigma_2, \sigma_3\rangle)$ after the 1 annotation, with respect to the branching choice.

c. We define a special component $W^{\#}$ for the case where the # character is annotated by 1:
   - **i.** The states of the component are: w^1, $w^1_{\mathbf{false}}$, $w^1_{\mathbf{true}}$, $w^{\#}$, w^{wait}, $w^{wait}_{\mathbf{false}}$ and $w^{wait}_{\mathbf{true}}$. The state w^1 is expecting to read 1, then it move to $w^{\#}$, that is expecting to read #, and then it move to w^{wait} to wait until the next 1 annotation is occurring. We use the same technique to deal with the annotation-reversing mechanism.
   - **ii.** The definition of the transitions between those states describes the behaviour specified earlier:
     - $\delta(w^1, 1) = w^1_{\mathbf{true}}$ and $\delta(w^1, 0) = w^1_{\mathbf{false}}$.
     - $\delta(w^1_{\mathbf{true}}, \checkmark) = \delta(w^1_{\mathbf{false}}, \sim) = w^{\#}$
     - $\delta(w^{\#}, \#) = w^{wait}$.
     - $\delta(w^{wait}, \sigma) = w^{wait}$ for every $\sigma \in \Sigma \setminus \{\#\}$.
     - $\delta(w^{wait}, 1) = w^{wait}_{\mathbf{false}}$ and $\delta(w^{wait}, 0) = w^{wait}_{\mathbf{true}}$.
     - $\delta(w^{wait}_{\mathbf{false}}, \sim) = \delta(w^{wait}_{\mathbf{true}}, \checkmark) = w^{wait}$.
     - $\delta(w^{wait}_{\mathbf{false}}, \checkmark) = \delta(w^{wait}_{\mathbf{true}}, \sim) = w^{\#}$.

d. We now describe the transitions of the initial state:
   - **i.** $\delta(w_{init}, \varepsilon) = w$ for every $w \in \{w^1\} \cup \bigcup_{x \in \Sigma \times (\Sigma \setminus \{\#\}) \times \Sigma \times \{d,l\}} \{b^x\}$. Those transitions represent guessing the position of the first 1 annotation.
   - **ii.** $\delta(w_{init}, \sigma) = w_{init}$ for every $\sigma \in \{l, r\} \cup \Sigma$. Those transitions represent waiting for the first 1 annotation to occur. We add the states $w^{init}_{\mathbf{false}}$ and $w^{init}_{\mathbf{true}}$ to allow unlimited 0 annotations, using the transitions:
     - $\delta(w_{init}, 0) = w^{init}_{\mathbf{true}}$ and $\delta(w_{init}, 1) = w^{init}_{\mathbf{false}}$.
     - $\delta(w^{init}_{\mathbf{true}}, \checkmark) = \delta(w^{init}_{\mathbf{false}}, \sim) = w_{init}$.

e. $W_{acc} = \{w_{acc}\} \cup \{e^x : x = (\sigma_1, \sigma_2, \sigma_3, d) \in \Sigma^3 \times \{l, r\}$ where $\sigma_1 \in (\{q_{acc}\} \times \Gamma)$ or $\sigma_2 \in (\{q_{acc}\} \times \Gamma)$ or $\sigma_3 \in (\{q_{acc}\} \times \Gamma)\}$ and we have that
$\delta(w, (q_{acc}, \gamma)) = w_{acc}$ for every $\gamma \in \Gamma$ and $w \in \{w_{init}, w^{wait}\} \cup \{b^x, e^x : x = (\sigma_1, \sigma_2, \sigma_3, d) \in \Sigma^3 \times \{l, r\}$ where $\sigma_1 \notin (\{q_{acc}\} \times \Gamma)$ and $\sigma_2 \notin (\{q_{acc}\} \times \Gamma)$ and $\sigma_3 \notin (\{q_{acc}\} \times \Gamma)$ and $d \in \{l, r\}\}$.

We continue to the case $t =$ MULTI and A_R is a DFW (or NFW). The reduction is similar: Let R' and A'_R be the regular language and the NFW described above, respectively. The only challenge is to construct a regular language R such that an instance of $\mathbf{true}^* \cdot R$ occurs iff an instance of R occurs, where R can be described by a DFW of size polynomial in $s(n)$.

This goal can be achieved with $R = 1 \cdot (0)^{s(n)} \cdot 0 + 1 \cdot (0)^k \cdot 1$, for every $k < s(n)$. Indeed, whenever the suffix of the computation is in R, Player 1 knows that the annotation of the last letter is incorrect. We can define an equivalent DFW $A_R = \langle AP, S, s_{init}, M, s_{acc}\rangle$ of size linear in $s(n)$ as follows.

1. The states set S contains the following states:
   - **a.** s_{init}, s_{rej} and s_{acc} which are the initial, rejecting and accepting states, respectively.
   - **b.** s_i for every $0 \le i \le s(n)$.
2. The transition function M defined as follows:

a. $M(s,\sigma) = s$ for every $s \in S \setminus \{s_{acc}\}$ and $\sigma \in \Sigma \cup \{\$, l, r\}$. $M(s,\sigma) = s_{rej}$ for $s \in \{a_{acc}, s_{rej}\}$ and for every $\sigma \in AP$.

b. $M(s_{init}, 1) = s_0$ and $M(s_{init}, 0) = s_{rej}$.

c. $M(s_i, 0) = s_{i+1}$ and $M(s_i, 1) = s_{acc}$ for every $0 \leq i \leq s(n) - 1$.

d. $M(s_{s(n)}, 0) = s_{acc}$ and $M(s_{s(n)}, 1) = s_{rej}$.