



Blinking Synthesis*

Orna Kupferman and Naama Shamash Halevy

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

Abstract

In *synthesis*, we automatically construct systems that satisfy their specifications. A reactive system interacts with its environment and has to satisfy its specification in all environments. Researchers have extensively studied synthesis in settings with partial visibility in which assignments to some input signals are hidden from the system throughout the interaction. We introduce and study *blinking synthesis* – synthesis in settings in which the assignments to all input signals are hidden in some rounds of the interaction. Blinking synthesis is relevant for systems with sensor failures, communication disruptions, or controlled switches.

The input to the blinking synthesis problem includes, in addition to a specification language L_φ , also a hiding language $L_\mathcal{H}$, describing behaviors that trigger hiding of input signals. The goal is to generate a system that satisfies L_φ in all environments that hide the assignment to input signals whenever the computation generated so far is in $L_\mathcal{H}$.

We show that while for LTL specifications the problem is 2EXPTIME-complete, a careful and parameterized analysis reveals it is harder than LTL synthesis: blinking synthesis is EXPTIME-complete for specifications given by deterministic automata and when one of the parameters, namely L_φ or $L_\mathcal{H}$, is fixed. We study settings with limited classes of $L_\mathcal{H}$ as well as ones in which the hiding language $L_\mathcal{H}$ can be synthesized too, aiming to either reduce sensing costs or maintain privacy of the environment.

1 Introduction

The rapid development of complex and safety-critical systems requires reliable *verification* methods. One of the most significant developments in this area has been the emergence of algorithmic methods for verifying *temporal specifications of reactive systems* [9]. A weakness of verification is that it is applied after the system has already been designed. More appealing are algorithmic methods for automatically generating correct-by-construction systems from their specifications. This is called *synthesis* [1, 8, 10, 25]. Synthesis enables designers to focus on *what* the system should do rather than *how* it should do it.

Early approaches to synthesis extracted a system from a proof that the specification is satisfiable. A reactive system interacts with its environment and should satisfy its specification in all environments [14, 1, 10]. The correct way to approach synthesis of reactive systems is to

*Research supported in part by the Israel Science Foundation, Grant 3177/25, and the European Research Council, Advanced Grant ADVANSYNT.

consider the situation as a (possibly infinite) *game* between the environment and the system. In each round in the game, the environment provides an assignment to a set I of input signals, and the system responds with an assignment to a set O of output signals. A specification φ over $I \cup O$ is then *realizable* if the system has a *strategy* to respond such that no matter how the environment behaves, the computation that the environment and the system generate together satisfies φ . The *synthesis problem* for φ is to return a finite-state system that realizes it (or an answer that φ is not realizable) [4].

Classical synthesis assumes *full observability*, where the system has access to the complete interaction history. More formally, a strategy for a system with inputs in I and outputs in O maps each finite word in $(2^I)^*$, which corresponds to a sequence of assignments to the input signals received so far, to an assignment to the output signals — the suggested action in 2^O for the system [25]. In practice, the system often has uncertainty about the behavior of the environment. This has led to the study of synthesis with *partial observability*, where strategies depend only on visible components of the history. Partial observability arises in at least two different forms. In *longitudinal observability*, some components of the environment are permanently hidden. In *transverse observability*, all components are observable, but only during selected rounds of the interaction [16].

Longitudinal observability corresponds to settings in which the environment has private variables. In particular, in distributed systems, each component typically has its own variables, and the specification may refer to all variables [26]. Transverse observability corresponds to settings in which observability is disabled along some rounds of the interaction. For example, by failure of communication links, deactivation of sensors, or switched systems where components are activated by a scheduler and are not aware of the input received while being switched off [12, 22, 23, 27]. Some models of partial observability combine aspects of longitudinal and transverse observability. For example, revealing POMDPs [3] allow the system to obtain complete information in some rounds, while in other rounds only partial information is available. Unlike our setting, the information revealed in these rounds is probabilistic and need not be completely hidden.

In the traditional *game-theoretic* approach, synthesis is reduced to deciding the winner in a two-player turn-based game between the system and the environment. The game is played on a graph whose vertices are partitioned between the system and the environment. The players generate an infinite path in the graph by jointly moving a token along the vertices of the graph. The token is initially placed on the initial vertex of the graph. Whenever the token is located at a vertex, its owner chooses the successor vertex. In games with full observability, the strategies of the players may depend on the history of the game so far. For synthesis of specifications in LTL, the vertices of the graph correspond to the states of a deterministic automaton for the specification. The objective of the system is that the jointly generated infinite path would satisfy the acceptance condition of the automaton, which corresponds to the interaction generating a computation that satisfies the specification.

Longitudinal observability corresponds to the system not knowing the exact action of the environment and hence not knowing the exact location of the token in the game graph [28, 21, 6, 7]. Transverse observability was studied in the context of *perspective games* [20]. There, the strategy of a player may depend only on the history of visits of the token in vertices the player owns. For example, the graph may model a communication network in which a company that owns part of the routers has to make routing decisions based only on information about visits to its routers [2]. Perspective games were studied also for non-zero sum games [19], and in a setting with notifications to the players [18].

Perspective games do not offer a solution to the synthesis problem in settings with transverse

observability. Indeed, in perspective games, once the system restores control and observability, it knows the current location of the token. In contrast, in synthesis, uncertainty about past observations translates into uncertainty about the current state of the specification automaton. The association of control with observability is at the heart of perspective games. In particular, it explains why the complexity of deciding perspective games is only polynomial in the game graph, even when the vertices of the graph are labeled by assignments to input and output signals, and the objective of the system is given by LTL formulas.

In this paper, we introduce and study *blinking synthesis*, where the assignments to all input signals are hidden from the system in some rounds during the interaction. Specifically, in addition to the specification $L_\varphi \subseteq (2^{I \cup O})^\omega$, the setting includes a *hiding language* $L_{\mathcal{H}} \subseteq (2^{I \cup O})^*$, describing the set of prefixes after which the assignment to the input signals is hidden. For example, $L_{\mathcal{H}}$ may contain all prefixes in which *enter_dark_zone* is assigned true with no *leave_dark_zone* assigned true after it. We assume that the system knows the language $L_{\mathcal{H}}$, and is aware of rounds in which the assignments to the input signals is hidden. Technically, in rounds of interactions after prefixes in $L_{\mathcal{H}}$, the system reads a special letter $\perp$. The system must satisfy the specification in all environments that provide $\perp$ rather than an assignment after exactly all prefixes in $L_{\mathcal{H}}$.

We start by analyzing the complexity of the blinking-synthesis problem in various settings. We assume that the hiding language $L_{\mathcal{H}}$ is given by a deterministic automaton that we refer to as a *blinker*. Our algorithms are based on reducing blinking synthesis to traditional synthesis by augmenting the an automaton for the specification with universal branches, which simulate all possible “fillings” for the hidden input assignments. Using this framework, we first prove that blinking synthesis for LTL specifications is 2EXPTIME-complete. While this matches the complexity of traditional LTL synthesis, it raises the question whether the doubly-exponential blow-up in the translation of LTL to deterministic automata dominates computational tasks that are unique to the blinking setting.

In order to answer this question, we provide a tighter analysis of the complexity, examining it for specifications given by deterministic automata. There, traditional synthesis is much easier (in particular, can be solved in PTIME for deterministic Büchi automata). We show that blinking synthesis is EXPTIME-complete for deterministic (or universal) parity and Büchi automata. We continue with a parameterized-complexity analysis and study its specification complexity, namely the complexity in terms of the specification, assuming the blinker is fixed, as well as its blinker complexity, namely the complexity in terms of the blinker, assuming the specification is fixed. We show that both are EXPTIME-complete. In particular, while traditional synthesis can be solved in constant time for fixed specifications, here an exponential dependency in the blinker cannot be avoided. Our most technically challenging results are the lower bounds for deterministic or fixed specifications, where the hidden information is carefully used for resolving nondeterminism or choices among exponentially many elements. Still in the computational front, we identify and study natural restrictions on the language $L_{\mathcal{H}}$, like temporal hiding, where visibility depends only on the number of rounds so far, or settings where $L_{\mathcal{H}}$ depends only on the input or only on the output signals, which affects the ability of the system to influence the hiding. We show that while most of our lower bounds hold also in these restricted setting, some restrictions make the problem easier. In particular, the blinker complexity when $L_{\mathcal{H}}$ only depends on output signals is PTIME.

Recall that we assume that the environment must hide input assignments after histories in $L_{\mathcal{H}}$. That is, $L_{\mathcal{H}}$ describes not a *possibility* that the input assignments are hidden but a *guarantee* that this is the case. We extend our complexity results to blinking synthesis in a stronger semantics, where $L_{\mathcal{H}}$ may over-approximate the prefixes that cause hiding. We show

that the latter semantics is indeed stronger, as it makes it more difficult for the system to learn about the missing information. Still, by manipulating the automata constructed for usual blinking synthesis, we show that all our complexity results hold also in the stronger semantics, as well as for a setting that allows both types (that is, possible and guaranteed) of hidings.

We continue and study settings in which the hiding language $L_{\mathcal{H}}$ is not given and should be synthesized with the system. We consider two incentives for the designer to increase $L_{\mathcal{H}}$. The first concerns *resources*: the system has a budget with which it can “buy” visibility in a given blinker. The second concerns *privacy*: in addition to a specification, the input to the synthesis problem includes *secrets*, and the synthesized hiding language should limit observability so that the system can still realize the specification yet the truth values of the secrets remain unknown. For the first case, we show that a naive search for visible states is optimal, in the sense that the problem is NP-complete for fixed specifications whose blinker complexity is PTIME. For the second case, we relate the challenge to synthesis with privacy in a more general setting, studied in [17]. There, hiding may be arbitrary – different sets of both input and output signals may be hidden in each round, and we show that blinking synthesis with privacy is not easier.

2 Preliminaries

2.1 Automata and LTL

For a finite *alphabet* Σ , a word $w = \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdots \in \Sigma^* \cup \Sigma^\omega$ is a (possibly infinite) sequence of letters in Σ . For an index $j \geq 1$, let $w[j]$ denote the j -th letter in w , thus $w[j] = \sigma_j$, and let $w[1, j]$ denote the prefix of length j of w , thus $w[1, j] = \sigma_1 \cdot \sigma_2 \cdots \sigma_j$. In particular, $w[1, 0] = \epsilon$. A *language* $L \subseteq \Sigma^*$ ($L \subseteq \Sigma^\omega$) is a set of finite (infinite) words.

An *automaton* is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Σ is an alphabet, Q is a finite set of *states*, $q_0 \in Q$ is an *initial state*, $\delta : Q \times \Sigma \rightarrow 2^Q$ is a *transition function*, and α is an *acceptance condition*, to be defined below. If $|\delta(q, \sigma)| \leq 1$ for every state $q \in Q$ and letter $\sigma \in \Sigma$, then $\mathcal{A}$ is *deterministic*. For a deterministic automaton $\mathcal{A}$ we view δ as a function $\delta : Q \times \Sigma \rightarrow Q$, where instead of writing $\delta(q, \sigma) = \{q\}$, we write $\delta(q, \sigma) = q$. The size $|\mathcal{A}|$ of the automaton $\mathcal{A}$ is the number its states, thus, $|\mathcal{A}| = |Q|$.

The automaton $\mathcal{A}$ may run on finite or infinite words. A *run* of $\mathcal{A}$ on an infinite word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states $r = r_0, r_1, r_2, \dots$, such that $r_0 = q_0$, and for all $i \geq 0$, we have that $r_{i+1} \in \delta(r_i, \sigma_{i+1})$. When $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_l \in \Sigma^*$ is finite, the run $r = r_0, r_1, r_2, \dots, r_l$ is finite too. The acceptance condition α determines which runs are “good”. For finite words, we have that $\alpha \subseteq Q$, and r satisfies α if its last state is in α . For infinite words, we consider here several acceptance conditions, all referring to the set $\text{inf}(r) \subseteq Q$ of states that r visits infinitely often. Formally, $\text{inf}(r) = \{q \in Q : r_i = q \text{ for infinitely many } i\}$. In *Büchi* automata, $\alpha \subseteq Q$, and r satisfies α if it visits states in α infinitely often, thus $\text{inf}(r) \cap \alpha \neq \emptyset$. Dually, in *co-Büchi* automata, $\alpha \subseteq Q$, and r satisfies α if it visits states in α only finitely often, thus $\text{inf}(r) \cap \alpha = \emptyset$. For $k \geq 1$, let $[k] = \{1, \dots, k\}$. In *parity* automata, $\alpha : Q \rightarrow [k]$ maps each state to a rank in $[k]$, for some $k \geq 1$, referred to as the *index* of α . A run r satisfies α if the maximal rank of a state in $\text{inf}(r)$ is even. Formally, $\max_{q \in \text{inf}(r)} \{\alpha(q)\}$ is even. A run that satisfies the acceptance condition α is *accepting*. A run that is not accepting is *rejecting*.

Unless $\mathcal{A}$ is deterministic, it may have several runs on a word. When $\mathcal{A}$ is *nondeterministic*, it accepts a word w if there is an accepting run of $\mathcal{A}$ on w . When $\mathcal{A}$ is *universal*, it accepts a word w if all its runs on w are accepting. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. Two automata are *equivalent* if their languages are equivalent.

We denote the different classes of automata by three-letter acronyms in $\{D, N, U\} \times$

$\{F, B, C, P\} \times \{W\}$. The first letter stands for the branching mode of the automaton (deterministic, nondeterministic, or universal); the second for the acceptance condition type (finite, Büchi, co-Büchi, or parity); and the third indicates we consider automata on words. For example, NBWs are nondeterministic Büchi word automata.

LTL is a linear temporal logic used for specifying on-going behaviors of reactive systems [24]. When a system is defined with respect to sets I and O of input and output signals, each state of the system induces an assignment in $2^{I \cup O}$, and each computation corresponds to a word in $(2^{I \cup O})^\omega$. Formulas of LTL are defined over the set $I \cup O$ using the usual Boolean operators and the temporal operators G (“always”) and F (“eventually”), X (“next time”) and U (“until”). The semantics of LTL is defined with respect to infinite computations in $(2^{I \cup O})^\omega$. Thus, each LTL formula φ over $I \cup O$ induces a language $L(\varphi) \subseteq (2^{I \cup O})^\omega$ of exactly all the computations that satisfy φ .

2.2 Synthesis

Let I and O be disjoint finite sets of input and output signals. We consider words over the alphabet $\Sigma = 2^{I \cup O}$ of truth assignments to the signals in $I \cup O$. For two words $w_I = i_1 \cdot i_2 \cdots \in (2^I)^\omega$ and $w_O = o_1 \cdot o_2 \cdots \in (2^O)^\omega$, we denote by $w_I \oplus w_O$ the word $(i_1 \cup o_1) \cdot (i_2 \cup o_2) \cdots \in \Sigma^\omega$ obtained by composing the assignments to the input and output signals. For simplicity, we sometimes refer to letters in Σ as pairs in $2^I \times 2^O$, replacing $i \cup o$ by $\langle i, o \rangle$.

We model systems that interact with their environments by (I/O) -transducers. An (I/O) -transducer is a tuple $\mathcal{T} = \langle I, O, S, s_0, \eta, \tau \rangle$, where I and O are sets of input and output signals, S is a finite set of states, $s_0 \in S$ is an initial state, $\eta : S \times 2^I \rightarrow S$ is a transition function, and $\tau : S \rightarrow 2^O$ is a labeling function. When I and O are clear from the context, we omit them, and simply write transducers. Consider an infinite sequence $w_I = i_1 \cdot i_2 \cdots \in (2^I)^\omega$ of assignments to input signals. The *run* of $\mathcal{T}$ on w_I is the infinite sequence of states $s_0, s_1, s_2, \dots$, where for all $j \geq 1$, we have that $s_j = \eta(s_{j-1}, i_j)$. Running on w_I , the transducer $\mathcal{T}$ generates an infinite sequence $\mathcal{T}(w_I) = o_1 \cdot o_2 \cdots \in (2^O)^\omega$ of assignments to the output signals, induced by the labels of the states that the run visits, starting from s_1 . That is, for all $j \geq 1$, we have that $o_j = \tau(s_j)$. Starting from s_1 (rather than s_0) reflects the fact that the environment initiates the interaction. The *computation* of $\mathcal{T}$ on w_I is then $w_I \oplus \mathcal{T}(w_I)$. The *language* of $\mathcal{T}$, denoted $L(\mathcal{T})$, is the set of computations of $\mathcal{T}$, that is $L(\mathcal{T}) = \{w_I \oplus \mathcal{T}(w_I) : w_I \in (2^I)^\omega\}$.

We say that $\mathcal{T}$ *realizes* a specification $L \subseteq \Sigma^\omega$, which describes desired behaviors, if for all $w_I \in (2^I)^\omega$, it holds that $w_I \oplus \mathcal{T}(w_I) \in L$. Equivalently, $\mathcal{T}$ realizes L if $L(\mathcal{T}) \subseteq L$. We say that a specification $L \subseteq \Sigma^\omega$ is *realizable* (with respect to a given partition of the signals to I and O) if there is an (I/O) -transducer that realizes it. In the *synthesis* problem, we are given a specification $L \subseteq \Sigma^\omega$, decide whether it is realizable, and return an (I/O) -transducer that realizes L , in case it is realizable.

2.3 Blinking synthesis

In *blinking synthesis*, the assignments to all input signals are hidden from the system in some rounds during the interaction. Specifically, in addition to the specification $L_\varphi \subseteq \Sigma^\omega$, the setting includes a *hiding language* $L_{\mathcal{H}} \subseteq \Sigma^*$, describing the set of prefixes of interactions after which the assignment to the input signals is hidden from the system. Let $L_{\mathcal{V}} = \Sigma^* \setminus L_{\mathcal{H}}$, thus $L_{\mathcal{V}}$ describes the set of prefixes of interactions after which the assignment to the input signal is visible to the system. Note that while only the assignments to input signals are hidden, the languages $L_{\mathcal{H}}$ and $L_{\mathcal{V}}$ are over $2^{I \cup O}$, thus membership in them depends on both input and output signals. We refer to $L_{\mathcal{V}}$ as the *visibility language*.

We assume that the system knows the language L_V , and is aware of rounds in which the assignment to the input signals is hidden. This corresponds to a synchronous setting, when the system and the environment move together according to a global clock, and thus the system expects an assignment and can figure out that information is hidden, for example when sensors return no data, visibility is blocked, or users hide their internal variables. Technically, we assume that instead of observing an assignment to the input signals, the systems reads a special letter $\perp$. More formally, in the blinking setting we model systems by *blinking (I/O)-transducers*, which differ from standard transducers in the input alphabet, which is now $2^I \cup \{\perp\}$. Thus, in each round, the system may either observe an assignment in 2^I or receive an indication that the current assignment is hidden. Formally, for sets of input and output signals I and O , let $\Sigma_{I,\mathcal{H}} = (2^I \cup \{\perp\})$ and $\Sigma_{\mathcal{H}} = (\Sigma_{I,\mathcal{H}} \times 2^O)$. We say that a letter $\langle i, o \rangle \in \Sigma_{\mathcal{H}}$ is a $\perp$ -letter if $i = \perp$. A blinking (I/O)-transducer $\mathcal{T} = \langle I, O, S, s_0, \eta, \tau \rangle$ differs from an (I/O)-transducer by the domain of η , which is now $S \times (2^I \cup \{\perp\})$. Note that a blinking transducer $\mathcal{T}$ defines a language over the alphabet $\Sigma_{\mathcal{H}}$, with $L(\mathcal{T}) = \{w_I \oplus \mathcal{T}(w_I) : w_I \in \Sigma_{I,\mathcal{H}}^\omega\}$.

For a word $w = \langle i_1, o_1 \rangle \cdot \langle i_2, o_2 \rangle \cdots \in \Sigma_{\mathcal{H}}^\omega$ and a hiding language $L_{\mathcal{H}} \subseteq \Sigma^*$, let $\text{hide}_{L_{\mathcal{H}}}(w) \in \Sigma_{\mathcal{H}}^*$ be the word obtained from w by changing the assignments to input signals after prefixes in $L_{\mathcal{H}}$ to $\perp$. That is, for all $j \geq 1$, if $w[1, j-1] \in L_{\mathcal{H}}$, then $\text{hide}_{L_{\mathcal{H}}}(w)[j] = \langle \perp, o_i \rangle$, and if $w[1, j-1] \notin L_{\mathcal{H}}$, then $\text{hide}_{L_{\mathcal{H}}}(w)[j] = w[j]$. Note that input assignments are hidden in rounds that are successive to rounds in which a prefix in $L_{\mathcal{H}}$ is generated.

For the reverse operation, we define, for a word $w \in \Sigma_{\mathcal{H}}^\omega$, the set of words $\text{fill}_{L_{\mathcal{H}}}(w) = \text{hide}_{L_{\mathcal{H}}}^{-1}(w)$. Note that $w' \in \Sigma^*$ is in $\text{fill}_{L_{\mathcal{H}}}(w)$, if w agrees with w' in all positions after prefixes that are not in $L_{\mathcal{H}}$, and that the replacement of $\perp$ by assignments in 2^I is such that whenever $j \geq 1$ is such that $w[j]$ is a $\perp$ -letter, then $w'[1, j-1] \in L_{\mathcal{H}}$. We expand $\text{fill}_{L_{\mathcal{H}}}$ to languages in the expected way. That is, for a language $L \subseteq \Sigma_{\mathcal{H}}^\omega$, we define $\text{fill}_{L_{\mathcal{H}}}(L) = \bigcup_{w \in L} \text{fill}_{L_{\mathcal{H}}}(w)$.

Example 2.1. Let $I = \{x\}$, $O = \{y\}$, and $L_{\mathcal{H}} = ((\bar{x}, \bar{y}) + (\bar{x}, y))^*$. That is, visibility is enabled forever after the environment assigns true to x . Consider the computation $w = (\bar{x}, \bar{y}) \cdot ((\bar{x}, y) \cdot (x, y))^\omega$. Then, $\text{hide}_{L_{\mathcal{H}}}(w) = (\perp, \bar{y}) \cdot (\perp, y) \cdot (\perp, y) \cdot ((\bar{x}, y) \cdot (x, y))^\omega$. Note that since visibility is enabled one round after prefixes in L_V and $w[1, 2]$ is in $L_{\mathcal{H}}$, we have that $\text{hide}_{L_{\mathcal{H}}}(w)[3]$ is a $\perp$ -letter even though $w[1, 3]$ is not in $L_{\mathcal{H}}$.

Now, for $w = (\perp, \bar{y}) \cdot (\perp, y) \cdot (\perp, y) \cdot ((\bar{x}, y) \cdot (x, y))^\omega$, there is only one word in $\text{fill}_{L_{\mathcal{H}}}^{-1}(w)$, namely $(\bar{x}, \bar{y}) \cdot ((\bar{x}, y) \cdot (x, y))^\omega$. Indeed, since $L_{\mathcal{H}} = ((\bar{x}, \bar{y}) + (\bar{x}, y))^*$, the only sequence of assignments to x that generates a word in which exactly the first three letters are $\perp$ -letters is $\bar{x}, \bar{x}, x$. Moreover, since $L_{\mathcal{H}}$ is prefix closed, then for every word $w \in \Sigma_{\mathcal{H}}^*$, if there is $j \geq 1$ such that $w[j]$ is not a $\perp$ -letter and $w[j+1]$ is a $\perp$ -letter, then $\text{fill}_{L_{\mathcal{H}}}^{-1}(w) = \emptyset$. Indeed, since $w[j+1]$ is a $\perp$ -letter, it must be that $w[1, j]$ is in $L_{\mathcal{H}}$, implying that $w[1, j-1]$ is in $L_{\mathcal{H}}$ too, and so $w[j]$ should be a $\perp$ -letter. In fact, it is not hard to see that for every word $w \in \Sigma_{\mathcal{H}}^\omega$, we have that $|\text{fill}_{L_{\mathcal{H}}}(w)| = 1$ if $w \in (\{\perp\} \times 2^O)^\omega + (\{\perp\} \times 2^O)^* \cdot \Sigma_{\mathcal{H}}^\omega$, and $|\text{fill}_{L_{\mathcal{H}}}(w)| = 0$ otherwise. $\square$

When a blinking transducer $\mathcal{T}$ interacts with an environment in the presence of input hiding, it has to satisfy the specification in all environments. For a specification $L_\varphi \subseteq \Sigma^\omega$ and a hiding language $L_{\mathcal{H}} \subseteq \Sigma^*$, we say that $\mathcal{T}$ *blinkingly realizes* $\langle L_\varphi, L_{\mathcal{H}} \rangle$ if for all $w_I \in \Sigma_{I,\mathcal{H}}^\omega$, we have that $\text{fill}_{L_{\mathcal{H}}}(w_I \oplus \mathcal{T}(w_I)) \subseteq L_\varphi$. Equivalently, $\text{fill}_{L_{\mathcal{H}}}(L(\mathcal{T})) \subseteq L_\varphi$. Note that there may be an input word $w_I \in \Sigma_{I,\mathcal{H}}^\omega$ for which $\text{fill}_{L_{\mathcal{H}}}(w_I \oplus \mathcal{T}(w_I)) = \emptyset$. In this case, by definition, $\text{fill}_{L_{\mathcal{H}}}(w_I \oplus \mathcal{T}(w_I)) \subseteq L_\varphi$ holds trivially. That is, if the environment hides input assignments after prefixes in L_V or does not hide them after prefixes in $L_{\mathcal{H}}$, then the system is not required to satisfy the specification. In Section 3.4, we study a stronger semantics to blinking synthesis, where $L_{\mathcal{H}}$ may over-approximate the prefixes that cause hiding and the input assignments after prefixes in $L_{\mathcal{H}}$ may not be hidden. Now, the pair $\langle L_\varphi, L_{\mathcal{H}} \rangle$ is *blinkingly realizable* if there is a blinking transducer that blinkingly realizes it.

In the *blinking-synthesis* problem, we are given a specification $L_\varphi \subseteq \Sigma^\omega$ and a hiding language $L_\mathcal{H} \subseteq \Sigma^*$, and we have to return a blinking transducer that blinkingly realizes $\langle L_\varphi, L_\mathcal{H} \rangle$, or decide that no such blinking transducer exists. Note that traditional synthesis coincides with blinking synthesis with $L_\mathcal{H} = \emptyset$.

The specification L_φ is typically given by an LTL formula over $I \cup O$ or an automaton over the alphabet $2^{I \cup O}$. The language $L_\mathcal{H}$ is given by a DFW, which we call a *blinker*. In order to simplify the presentation, we describe a blinker as a DFW with two complementing acceptance conditions, thus $\mathcal{B} = \langle \Sigma, Q, q_0, \delta, \mathcal{V}, \mathcal{H} \rangle$ is such that $\mathcal{V}$ and $\mathcal{H}$ form a partition of Q . We use $\mathcal{B}_\mathcal{V}$ and $\mathcal{B}_\mathcal{H}$ to denote the DFWs obtained from $\mathcal{B}$ by letting $\mathcal{V}$ and $\mathcal{H}$, respectively, be the set of accepting states. The DFWs $\mathcal{B}_\mathcal{V}$ and $\mathcal{B}_\mathcal{H}$ then define the languages $L_\mathcal{V}$ and $L_\mathcal{H}$, respectively. Thus, input assignments are visible after prefixes that reach $\mathcal{V}$ in $\mathcal{B}$, and are hidden after prefixes that reach $\mathcal{H}$.

We sometimes replace L_φ and $L_\mathcal{H}$ in the different notations with automata or LTL formulas that describe them, thus talk about blinking realizability of $\langle \mathcal{A}_\varphi, \mathcal{B} \rangle$ or $\langle \varphi, \mathcal{B} \rangle$, for an automaton $\mathcal{A}_\varphi$, an LTL formula φ , and a blinker $\mathcal{B}$. Likewise, we use $\text{fill}_\mathcal{B}$ and $\text{hide}_\mathcal{B}$ in order to refer $\text{fill}_{L_\mathcal{H}}$ and $\text{hide}_{L_\mathcal{H}}$, with $L_\mathcal{H} = L(\mathcal{B}_\mathcal{H})$.

Example 2.2. Let $I = \{x\}$, $O = \{y\}$, and $L_\mathcal{H} = ((\bar{x}, \bar{y}) + (\bar{x}, y))^*$. As discussed in Example 2.1, for every words $w \in \Sigma_\mathcal{H}^\omega$, we have that $|\text{fill}_{L_\mathcal{H}}(w)| \leq 1$. Moreover, as long as the input assignment is hidden from the system, it can infer that the assignments to x so far are of the form $(\bar{x})^* \cdot (x + \bar{x})$, and once $\perp$ is replaced by an assignment $b \in \{x, \bar{x}\}$, it can infer that the assignments to x so far are of the form $(\bar{x})^* \cdot x \cdot b$. The system may use this inference. For example, the blinking transducer in Figure 1 blinkingly realizes the specification $\varphi_1 = \text{G}(x \leftrightarrow \text{X}y)$: as long as it observes $\perp$, it infers that the assignment to x in the previous round was $\bar{x}$, and it outputs $\bar{y}$. Once it observes an assignment $b \in \{x, \bar{x}\}$, it knows that the assignment to x in the previous round was x , and that no further hides are expected, and thus moves to a state space that remembers the last two assignments to x .

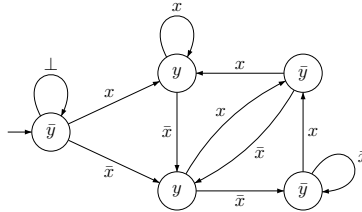


Figure 1: A blinking transducer that blinkingly realizes $\langle \text{G}(x \leftrightarrow \text{X}y), ((\bar{x}, \bar{y}) + (\bar{x}, y))^* \rangle$.

On the other hand, for $\varphi_2 = \text{G}(x \leftrightarrow y)$, the pair $\langle \varphi_2, L_\mathcal{H} \rangle$ is not blinkingly realizable. Indeed, now the system should react according to the last assignment to x , which cannot be inferred. Note, however, that if $L_\mathcal{H}$ is such that $(\{x, y\}, \{\bar{x}, \bar{y}\})^* \cap L_\mathcal{H} = \emptyset$, then $\langle \varphi_2, L_\mathcal{H} \rangle$ is blinkingly realizable. Indeed, as long as the system copies the value of x to y , it satisfies $x \leftrightarrow y$, the blinker stays in a state in $\mathcal{V}$, and the system can continue to satisfy $x \leftrightarrow y$. $\square$

3 The Complexity of the Blinking-Synthesis Problem

In this section we study the complexity of the blinking-synthesis problem. We start with LTL blinking synthesis, where the complexity coincides with the 2EXPTIME complexity of traditional LTL synthesis. As discussed in Section 1, in both the traditional and blinking

settings, the doubly-exponential complexity follows from the need to translate LTL formulas to deterministic automata, which may dominate and hide components of the algorithm that exist only in the blinking setting. In order to reveal such components, we continue and examine the complexity more carefully, analyzing it for specifications given by deterministic automata, and providing a parameterized-complexity analysis, thus when the specification or blinker are fixed. We show that blinking synthesis is EXPTIME-complete in all these cases, revealing it is harder than traditional synthesis. We consider settings where the hiding language is restricted, for example depends only on time, and study how the different restrictions influence the complexity of the problem. Finally, we extend the complexity analysis to a stronger semantics of blinking synthesis, where the assignment after prefixes in $L_{\mathcal{H}}$ may not be hidden.

3.1 LTL blinking synthesis

In this section we develop the main tool with which we solve the blinking-synthesis problem, which is a reduction of blinking synthesis to traditional synthesis. From the technical point of view, the reduction involves adding universal branches to an automaton for the specification. The complexity analysis of various formalisms and restrictions then follows from a careful examination of how the addition of the universal branches influences the type of automaton to which traditional synthesis is eventually applied. Essentially, universal branches are added in both the specification automaton and the blinker, and interesting cases involve setting in which the specification automaton has universal branches in the first place, as well as visibility languages in which such branches can be avoided.

We start by describing the reduction from blinking to traditional synthesis. In the reduction, we view a blinking transducer $\mathcal{T}$ as a traditional transducer, thus refer to $\perp$ as a special assignment to the input signals and to $\mathcal{T}$ as a transducer that receives from the environment words over the alphabet $\Sigma_{I,\mathcal{H}}$ and aims to satisfy a specification over $\Sigma_{\mathcal{H}}$.

For a specification $L_{\varphi} \subseteq \Sigma^{\omega}$ and a hiding language $L_{\mathcal{H}} \subseteq \Sigma^*$, let

$$\text{blink}(L_{\varphi}, L_{\mathcal{H}}) = \{w \in \Sigma_{\mathcal{H}}^{\omega} : \text{fill}_{L_{\mathcal{H}}}(w) \subseteq L_{\varphi}\}.$$

That is, $\text{blink}(L_{\varphi}, L_{\mathcal{H}})$ contains all words over $\Sigma_{\mathcal{H}}^{\omega}$ all whose fillings that are consistent with $L_{\mathcal{H}}$ are in L_{φ} . As with the other notations, we sometimes replace L_{φ} by the LTL formula or automaton that defines it and replace $L_{\mathcal{H}}$ by the blinker that defines it and talk about the languages $\text{blink}(\varphi, \mathcal{B})$ or $\text{blink}(\mathcal{A}, \mathcal{B})$.

Theorem 3.1. *Consider a specification $L_{\varphi} \subseteq \Sigma^{\omega}$ and a hiding language $L_{\mathcal{H}} \subseteq \Sigma^*$. Then, $\langle L_{\varphi}, L_{\mathcal{H}} \rangle$ is blinkingly realizable iff $\text{blink}(L_{\varphi}, L_{\mathcal{H}})$ is realizable. Moreover, a blinking transducer $\mathcal{T}$ blinkingly realizes $\langle L_{\varphi}, L_{\mathcal{H}} \rangle$ iff $\mathcal{T}$ realizes $\text{blink}(L_{\varphi}, L_{\mathcal{H}})$.*

Proof. The proof follows easily from the definition. Consider a blinking transducer $\mathcal{T}$. By definition, $\mathcal{T}$ blinkingly realizes $\langle L_{\varphi}, L_{\mathcal{H}} \rangle$ iff for every word $w_I \in \Sigma_{I,\mathcal{H}}$, we have that $\text{fill}_{L_{\mathcal{H}}}(w_I \oplus \mathcal{T}(w_I)) \subseteq L_{\varphi}$, which holds iff $w_I \oplus \mathcal{T}(w_I) \in \text{blink}(L_{\varphi}, L_{\mathcal{H}})$. Equivalently, $\mathcal{T}$ viewed as a traditional transducer realizes $\text{blink}(L_{\varphi}, L_{\mathcal{H}})$. $\square$

Reasoning about the language $\text{blink}(L_{\varphi}, L_{\mathcal{H}})$ involves reasoning about all the words in $\text{fill}_{L_{\mathcal{H}}}(w)$, for each word $w \in \Sigma_{\mathcal{H}}^{\omega}$. Such a reasoning can be naturally done by universal automata, with copies sent for all possible fillings of w . Formally, we have the following.

Lemma 3.2. *Given a UCW $\mathcal{U}$ over the alphabet Σ and a blinker $\mathcal{B}$, we can construct a UCW $\mathcal{U}_{\text{blink}}^{\mathcal{B}}$ over the alphabet $\Sigma_{\mathcal{H}}$ such that $\mathcal{U}_{\text{blink}}^{\mathcal{B}}$ is of size $O(|\mathcal{U}| \cdot |\mathcal{B}|)$ and $L(\mathcal{U}_{\text{blink}}^{\mathcal{B}}) = \text{blink}(\mathcal{U}, \mathcal{B})$.*

Proof. Let $\mathcal{U} = \langle \Sigma, S, s_0, \delta_\varphi, \alpha \rangle$ and $\mathcal{B} = \langle \Sigma, Q, q_0, \delta_\mathcal{B}, \mathcal{V}, \mathcal{H} \rangle$. We define $\mathcal{U}_{\text{blink}}^\mathcal{B}$ so that when it runs on a word $w \in \Sigma_\mathcal{H}^\omega$, it sends copies that correspond to the runs of $\mathcal{U}$ on words in $\text{fill}_\mathcal{B}(w)$. A word w is accepted by $\mathcal{U}_{\text{blink}}^\mathcal{B}$ iff all these runs of $\mathcal{U}$ are accepting, which holds iff $w \in \text{blink}(\mathcal{U}, \mathcal{B})$.

Formally, $\mathcal{U}_{\text{blink}}^\mathcal{B} = \langle \Sigma_\mathcal{H}, (S \times Q) \cup \{q_{\text{acc}}\}, \langle s_0, q_0 \rangle, \delta', \alpha \times Q \rangle$, where q_{acc} is an accepting sink, and δ' is defined, for every $\langle s, q \rangle \in S \times Q$ and $\langle i, o \rangle \in \Sigma_\mathcal{H}$, as follows.

$$\delta'(\langle s, q \rangle, \langle i, o \rangle) = \begin{cases} \delta_\varphi(s, \langle i, o \rangle) \times \{\delta_\mathcal{B}(q, \langle i, o \rangle)\} & \text{if } q \in \mathcal{V} \text{ and } i \neq \perp \\ \bigcup_{i' \in 2^I} \delta_\varphi(s, \langle i', o \rangle) \times \{\delta_\mathcal{B}(q, \langle i', o \rangle)\} & \text{if } q \in \mathcal{H} \text{ and } i = \perp \\ \{q_{\text{acc}}\} & \text{otherwise} \end{cases}$$

Recall that each run of $\mathcal{U}_{\text{blink}}^\mathcal{B}$ on w corresponds to a run of $\mathcal{U}$ on a word obtained by filling the missing inputs in w . Runs corresponding to words that are not in $\text{fill}_\mathcal{B}(w)$ reach q_{acc} and are thus accepting. Runs corresponding to words in $\text{fill}_\mathcal{B}(w)$ are accepting iff the word is accepted by $\mathcal{U}$, thus is in L_φ . It is easy to see that $\mathcal{U}_{\text{blink}}^\mathcal{B}$ has $|S| \cdot |Q| + 1$ states. In the full version, we prove formally that $L(\mathcal{U}_{\text{blink}}^\mathcal{B}) = \text{blink}(\mathcal{U}, \mathcal{B})$. $\square$

The important feature in working with universal automata is that handling of hidden information leaves us with universal automata, maintaining the exponential translation from LTL formulas. This allows us to solve LTL blinking realizability in 2EXPTIME. Thus, the problem is not harder than traditional LTL synthesis. Formally, we have the following.

Theorem 3.3. *LTL blinking synthesis is 2EXPTIME-complete.*

Proof. We start with the upper bound. Consider an LTL formula φ and a blinker $\mathcal{B}$. Let $\mathcal{U}_{\neg\varphi}$ be an NBW recognizing $L(\neg\varphi)$. By [30], there is such an NBW of size $2^{O(|\varphi|)}$. By dualizing $\mathcal{U}_{\neg\varphi}$, we obtain a UCW $\mathcal{U}_\varphi$ recognizing $L(\varphi)$. Now, applying the construction in Lemma 3.2 to $\mathcal{U}_\varphi$ and $\mathcal{B}$, we get a UCW $\mathcal{U}_{\text{blink}}^\mathcal{B}$ of size $2^{O(|\varphi|)} \cdot |\mathcal{B}|$ that recognizes the language $\text{blink}(\varphi, \mathcal{B})$. By Theorem 3.1, blinking synthesis of $\langle \varphi, \mathcal{B} \rangle$ can be reduced to synthesis of $\mathcal{U}_{\text{blink}}^\mathcal{B}$. Since synthesis of a UCW with n states can be solved in time $2^{O(n)}$ [15], the 2EXPTIME upper bound follows from $\mathcal{U}_{\text{blink}}^\mathcal{B}$ having $2^{O(|\varphi|)} \cdot |\mathcal{B}|$ states.

For the lower bound, note that an LTL formula φ is realizable iff $\langle \varphi, \emptyset \rangle$ is blinkingly realizable. Since LTL realizability is 2EXPTIME-hard [29], we are done. $\square$

3.2 A tighter complexity analysis

In this section we analyze the complexity of blinking synthesis more carefully. The technical challenge is in our lower bounds, where we show that blinking synthesis is more complex than traditional synthesis. We start with specifications given by universal or deterministic automata. This setting is of interest, as traditional synthesis for DBWs can be solved in PTIME, and, as we show below, is exponentially harder for blinking synthesis. Intuitively, the hidden information can encode runs, making the handling of deterministic automata as hard as that of universal automata.

Theorem 3.4. *UPW blinking realizability is EXPTIME-complete. Hardness in EXPTIME applies already for DBW blinking realizability.*

Proof. Given a UPW $\mathcal{U}_\varphi$ and a blinker $\mathcal{B}$, let $\mathcal{U}_{\text{blink}}^\mathcal{B}$ be a UPW obtained from $\mathcal{U}_\varphi$ as in Lemma 3.2, now applied to parity automata. That is, the rank of a state $\langle s, q \rangle$ in $\mathcal{U}_{\text{blink}}^\mathcal{B}$ is the rank of s in $\mathcal{U}_\varphi$. Note that the correctness of the construction in the proof of Lemma 3.2 is independent of the type of the acceptance condition, thus $L(\mathcal{U}_{\text{blink}}^\mathcal{B}) = \text{blink}(\mathcal{U}_\varphi, \mathcal{B})$. By Theorem 3.1, blinking

synthesis of $\langle \mathcal{U}_\varphi, \mathcal{B} \rangle$ can be reduced to synthesis of $\text{blink}(\mathcal{U}_\varphi, \mathcal{B})$. Since $\mathcal{U}_{\text{blink}}^\mathcal{B}$ is of size $|\mathcal{U}_\varphi| \cdot |\mathcal{B}|$, and synthesis for UPWs can be solved in exponential time [11], we are done.

For the lower bound, we describe a polynomial-time reduction from UBW realizability, which is EXPTIME-hard [15], to DBW blinking realizability. Given a UBW $\mathcal{U}$ over $2^{I \cup O}$, we define a set of signals I' , a DBW $\mathcal{D}_\varphi$ over $2^{I' \cup O}$, and a two-state blinker $\mathcal{B}$, such that $|\mathcal{D}_\varphi|$ is polynomial in $|\mathcal{U}|$ and $\mathcal{U}$ is realizable iff $\langle \mathcal{D}_\varphi, \mathcal{B} \rangle$ is blinkingly realizable.

The idea of the construction is as follows. The blinker $\mathcal{B}$ defines the visibility language to include exactly all words of odd length. The signals in I' include, in addition to these in I , signals that encode the state space of $\mathcal{U}$. Each letter in $2^{I' \cup O}$ thus encode both a letter in $2^{I \cup O}$ and a state in Q . When the DBW $\mathcal{D}_\varphi$ reads a word w , it cares about the letters in $2^{I \cup O}$ in the odd positions of w and about the states in Q in the even positions of w . It accepts w if the sequence of states in the even position is an accepting run of $\mathcal{U}$ on the word in the odd positions. Not observing the assignment in even positions has the effect of viewing only the letters, without viewing a run on them, which captures the difference between deterministic and universal automata. Consequently, a blinking transducer for $\mathcal{D}_\varphi$ that observes only letters in odd positions corresponds to a transducer that aims at realizing $\mathcal{U}$.

We now describe the reduction formally. Let $\mathcal{U} = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$, let I_Q be a set of $\lceil \log |Q| \rceil$ signals that encode Q , and let $I' = I \cup I_Q$. Also, let $\Sigma = 2^{I \cup O}$ and $\Sigma' = 2^{I' \cup O}$. For a letter $\sigma' \in \Sigma'$, we denote its Q -component by $\sigma'|_Q$, and its Σ -component by $\sigma'|\Sigma$. We extend these notations to words: for $w' = \sigma'_1 \cdot \sigma'_2 \cdots \in \Sigma'^\omega$, we have that $w'|_Q = \sigma'_1|_Q \cdot \sigma'_2|_Q \cdots$ and $w'|\Sigma = \sigma'_1|\Sigma \cdot \sigma'_2|\Sigma \cdots$. We view a word $w = (i_1 \cup o_1) \cdot (i_2 \cup o_2) \cdots \in (\Sigma')^\omega$ as the combination $w_{\text{odd}} \odot w_{\text{even}}$ of a word $w_{\text{odd}} = (i_1 \cup o_1) \cdot (i_3 \cup o_3) \cdots \in (2^{I \cup O})^\omega$, obtained by concatenating the letters in odd positions in w , with a word $w_{\text{even}} = (i_2 \cup o_2) \cdot (i_4 \cup o_4) \cdots \in (2^{I' \cup O})^\omega$, achieved by concatenating the letters in even positions.

We define $\mathcal{D}_\varphi$ so that $L(\mathcal{D}_\varphi) = \{w \in (\Sigma')^\omega : \text{the sequence } w_{\text{even}}|_Q \text{ is an accepting run of } \mathcal{U} \text{ on the word } w_{\text{odd}}|\Sigma\}$. Note that for every word $w \in L(\mathcal{U})$, it holds that $w \odot r \in L(\mathcal{D}_\varphi)$ for every r for which $r|_Q$ describes a run of $\mathcal{U}$ on w . Also, for every word $w \notin L(\mathcal{U})$, there is a rejecting run r of $\mathcal{U}$ on w , thus $w \odot s \notin L(\mathcal{D}_\varphi)$ for every s with $s|_Q = r$.

We define $\mathcal{D}_\varphi = \langle \Sigma', Q', q_0, \delta', \alpha \rangle$ as follows. First, $Q' = Q \cup (Q \times \Sigma) \cup \{q_{\text{acc}}\}$, where q_{acc} is an accepting sink. Then, the transition function $\delta' : Q' \times \Sigma' \rightarrow Q'$ uses the Q -component of letters in even positions in order to resolve the universal requirements in $\mathcal{U}$. Formally, this is done as follows. For every state $q \in Q$ and letter $\sigma' \in \Sigma'$, we have that $\delta'(q, \sigma') = \langle q, \sigma'|\Sigma \rangle$. That is, from states in Q , the transition function ignores signals in I_Q and leads to a state that remembers the current state and the letter encoded in σ' . Now, for every state $\langle q, \sigma \rangle \in Q \times \Sigma$ and letter $\sigma' \in \Sigma'$, we have that $\delta'(\langle q, \sigma \rangle, \sigma')$ is $\sigma'|_Q$ if $\sigma'|_Q \in \delta(q, \sigma)$, and is q_{acc} otherwise. That is, in a state $\langle q, \sigma \rangle \in Q \times \Sigma$, we ignore signals in I and consider the state encoded in σ' . If this state is not a successor of q with σ , then we move to an accepting sink. Otherwise, we proceed to this state. Note that if the state component of a word over Σ' is always a transition of δ , then the obtained word is of the form $w \oplus r$, where $r|_Q$ is a run of $\mathcal{U}$ on $w|_\Sigma$, and is accepted iff $r|_Q$ is accepting. Note also that for every $w \in \Sigma'^\omega$ such that $w|_\Sigma \in L(\mathcal{U})$, we have that $w \oplus s \in L(\mathcal{D}_\varphi)$ for every $s \in (\Sigma')^\omega$. Indeed, if $s|_Q$ is a valid run of $\mathcal{U}$ on $w|_\Sigma$, then it satisfies α , and so the run of $\mathcal{D}_\varphi$ on $w \oplus s$ is accepting. Otherwise, the run of $\mathcal{D}_\varphi$ on $w \oplus s$ reaches q_{acc} , and is also accepting.

We now prove formally that $\mathcal{U}$ is realizable iff $\langle \mathcal{D}_\varphi, \mathcal{B} \rangle$ is blinkingly realizable. For the first direction, assume that $L(\mathcal{U})$ is realizable. Thus, there is a transducer $\mathcal{T} = \langle I, O, S, s_0, \eta, \tau \rangle$ that realizes it. A blinking transducer $\mathcal{T}'$ that blinkingly realizes $\langle \mathcal{D}_\varphi, \mathcal{B} \rangle$ can follow $\mathcal{T}$ in odd positions. Formally, $\mathcal{T}' = \langle I' \cup \{\perp\}, O, S', s_0, \eta', \tau' \rangle$, where $S' = S \cup \{s' : s \in S\}$, $\tau'(s') = \tau(s)$ for all $s \in S$, $\tau'(s) = o$ for every $s \in S$ and an arbitrary $o \in O$, $\eta'(s, i) = t'$ for every $s \in S$

and $i \in 2^{I'}$ such that $t = \eta(s, i|_\Sigma)$, and $\eta'(s', i) = s$ for every $s \in S$ and $i \in 2^{I'}$. That is, when reading a word $i_1 \cdot \perp \cdot i_2 \cdot \perp \cdots \in (I' \cup \{\perp\})^\omega$, the transducer $\mathcal{T}'$ outputs $o_1 \cdot o \cdot o_2 \cdot o \cdots$, where $o_1 \cdot o_2 \cdots = \mathcal{T}(i_1|_\Sigma \cdot i_2|_\Sigma \cdots)$. We claim that $\mathcal{T}'$ blinkingly realizes $\langle \mathcal{D}_\varphi, \mathcal{B} \rangle$. Indeed, for every $w \in L(\mathcal{T}')$, we have that $w_{\text{odd}}|_\Sigma \in L(\mathcal{T})$. Since $\mathcal{T}$ realizes $L(\mathcal{U})$, we get that $w_{\text{odd}}|_\Sigma \in L(\mathcal{U})$. Thus, $w \oplus s \in L(\mathcal{D}_\varphi)$ for every $s \in \Sigma'^\omega$, and in particular for $s = w|_{\text{even}}$.

For the second direction, let $\mathcal{T} = \langle I' \cup \{\perp\}, O, S, s_0, \eta, \tau \rangle$ be a blinking transducer that blinkingly realizes $\langle \mathcal{D}_\varphi, \mathcal{B} \rangle$, and consider the transducer $\mathcal{T}' = \langle I, O, S \cup \{s'\}, s', \eta', \tau' \rangle$, where τ' is defined as τ on S and $\tau'(s') = \tau(s_0)$, and $\eta'(s, i) = \eta^*(s, \perp \cdot i)$ for every $s \in S$ and $i \in I$, and $\eta'(s', i) = \eta(s_0, i)$ for every $i \in I$. Thus, when reading a word $i_1 \cdot i_2 \cdots \in I^\omega$, the transducer $\mathcal{T}'$ outputs a word $o_1 \cdot o_2 \cdots$ such that $(i_1 \cup o_1) \cdot (\{\perp\} \cup o'_1) \cdot (i_2 \cup o_2) \cdot (\{\perp\} \cup o'_2) \cdots \in L(\mathcal{T})$ for some $o'_1 \cdot o'_2 \cdots \in O^\omega$. Since $\mathcal{T}$ blinkingly realizes $\langle \mathcal{D}_\varphi, \mathcal{B} \rangle$, it holds that for all $i'_1 \cdot i'_2 \cdots \in I'^\omega$, the word $(i_1 \cup o_1) \cdot (i'_1 \cup o'_1) \cdot (i_2 \cup o_2) \cdot (i'_2 \cup o'_2) \cdots$ is in $L(\mathcal{D}_\varphi)$. Thus, for every run r of $\mathcal{U}$ on the word $w = (i_1 \cup o_1) \cdot (i_2 \cup o_2) \cdots$, the word $w \oplus r$ is in $L(\mathcal{D}_\varphi)$, which implies $w \in L(\mathcal{U})$, and we are done. $\square$

We continue to a *parameterized-complexity* analysis of blinking synthesis. Specifically, we study the *specification complexity* of blinking synthesis, namely its complexity in terms of the specification, assuming the blinker is of a fixed size, as well as its *blinker complexity*, namely the complexity in terms of the blinker, assuming the specification is fixed. We start with the case the blinker is fixed, where things are easy.

Theorem 3.5. *The specification complexities of LTL and DBW blinking synthesis are 2EXPTIME-complete and EXPTIME-complete, respectively.*

Proof. The upper bounds follow from the joint complexity, proven in Theorems 3.3 and 3.4. For the lower bounds, note that the lower bound in the proof of Theorem 3.3 uses $L_{\mathcal{H}} = \emptyset$, which can be defined by a single-state blinker, and the lower bound in the proof of Theorem 3.4 uses $L_{\mathcal{H}} = (\Sigma \cdot \Sigma)^*$, which can be defined by a two-state blinker. $\square$

We continue to blinking synthesis of specifications of a fixed size. Clearly, traditional synthesis for such specifications can be solved in a constant time. For blinking synthesis, we show that an exponential complexity in the blinker cannot be avoided. Note that since the specification is fixed, its formalism does not influence the complexity.

Theorem 3.6. *The blinker complexity of blinking synthesis is EXPTIME-complete.*

Proof. For the upper bound, observe that when we start with an LTL formula of a fixed size, the UCW that corresponds to it is also of a fixed size, and so the size of the UCW $\mathcal{U}_{\text{blink}}^{\mathcal{B}}$ constructed in Lemma 3.2 is $O(|\mathcal{B}|)$, and the synthesis problem for it can be solved in EXPTIME.

For the lower bound, we describe a reduction from the problem of deciding the nonemptiness of an intersection of finite deterministic tree automata, which proved to be EXPTIME-hard in [13]. We first need some definitions. A deterministic automaton on finite trees (DFT) is $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\alpha \subseteq Q$ is a set of accepting states, and $\delta : Q \times \Sigma \rightarrow Q \times Q$ is a transition function. For convenience, we denote by δ_l and δ_r the left and right components of δ , respectively. That is, for all $q \in Q$ and $\sigma \in \Sigma$, we have that $\delta(q, \sigma) = \langle \delta_l(q, \sigma), \delta_r(q, \sigma) \rangle$. Given a tree $T = \langle \eta, \iota \rangle$, where $\eta \subseteq \{0, 1\}^*$, is a prefix-closed set of nodes and $\iota : \eta \rightarrow \Sigma$ labels each node by a letter in Σ , the run of $\mathcal{A}$ on T is a Q -labeled tree $\langle \eta, \iota_{\mathcal{A}} \rangle$, where $\iota_{\mathcal{A}}(\epsilon) = q_0$, and for every $x \in \eta$ such that x is not a leaf, we have that $\langle \iota_{\mathcal{A}}(x \cdot 0), \iota_{\mathcal{A}}(x \cdot 1) \rangle = \delta((\iota_{\mathcal{A}}(x), \iota(x)))$. A run is accepting if every leaf is labeled by a state in α . A Σ -labeled tree T is accepted by $\mathcal{A}$ if the run of $\mathcal{A}$ on T

is accepting. The nonempty-intersection problem gets as input DFTs $\mathcal{A}_1, \dots, \mathcal{A}_n$, and decides whether their intersection is nonempty, that is $\bigcap_{k \in [n]} L(\mathcal{A}_k) \neq \emptyset$.

We now describe the reduction. Let I and O be such that $x \in I$ and $free, y \in O$, and consider the fixed LTL specification $\varphi = \text{XF}free \wedge \text{FG}(x \leftrightarrow y)$. Given DFTs $\mathcal{A}_1, \dots, \mathcal{A}_n$, we construct a blinker $\mathcal{B}$ such that $\bigcap_{k \in [n]} L(\mathcal{A}_k) \neq \emptyset$ iff $\langle \varphi, \mathcal{B} \rangle$ is blinkingly realizable. Note that in order to satisfy $\text{FG}(x \leftrightarrow y)$, the assignment to x must be hidden only finitely often, thus the blinker should eventually enable visibility forever. In the beginning of the interaction, visibility is disabled, and the environment chooses $k \in [n]$. Then, visibility is enabled, and the system should keep the value of $free$ false until it convinces the blinker that it can describe a tree in $L(\mathcal{A}_k)$. Since the system does not know k , it can do this iff $\bigcap_{k \in [n]} L(\mathcal{A}_k) \neq \emptyset$, in which case the system can describe a tree in the intersection. Once the description of the tree completes successfully, the system outputs $free$ and the blinker moves to a sink in which visibility is enabled, which enables the system to satisfy $\text{FG}(x \leftrightarrow y)$.

Challenging the system to generate a tree in $L(\mathcal{A}_k)$ is done in an area of $\mathcal{B}$ where input assignments are visible (i.e, states in $\mathcal{V}$): at each round, the environment provides a direction in $\{0, 1\}$, and the system responds with a letter in Σ . If the run of $\mathcal{A}_k$ on the generated path reaches α , the system completes the mission of generating a tree in $L(\mathcal{A}_k)$, and the blinker moved to the sink in which visibility is enabled. In addition, the system keep the value of $free$ false during this phase.

In the full version, we describe $\mathcal{B}$ in detail and prove formally that $\bigcap_{k \in [n]} L(\mathcal{A}_k) \neq \emptyset$ iff $\langle \varphi, \mathcal{B} \rangle$ is blinkingly realizable. $\square$

3.3 Special cases of hiding languages

While hiding languages may be arbitrary regular languages, several special cases capture natural hiding policies that arise in practice. In this section we identify and study such cases and examine their implication on the complexity of blinking synthesis.

We first need some definition. Consider a blinker $\mathcal{B} = \langle 2^{I \cup O}, Q, q_0, \delta, \mathcal{V}, \mathcal{H} \rangle$, a set of signals $P \subseteq I \cup O$, and a set of states $S \subseteq Q$. We say that $\mathcal{B}$ is (P, S) -independent if the transitions from all states in S depend only on signals in $(I \cup O) \setminus P$. Formally, for every state $s \in S$, assignment $\sigma \in 2^{I \cup O}$, and signal $p \in P$, we have that $\delta(s, \sigma \cup \{p\}) = \delta(s, \sigma \setminus \{p\})$. Intuitively, a (P, S) -independent blinker does not need to read assignments to signals in P from states in S in order to decide how to proceed. Note that if $\mathcal{B}$ is (P, S) -independent, then it is also (P', S') -independent for every $P' \subseteq P$ and $S' \subseteq S$. When $S = Q$, we say that $\mathcal{B}$ is P -independent.

We consider the following types of a hiding language $L_{\mathcal{H}} \subseteq \Sigma^*$.

- *Temporal hiding*, when $L_{\mathcal{H}}$ is defined by an $(I \cup O)$ -independent blinker.
- *System-independent hiding*, when $L_{\mathcal{H}}$ is defined by an O -independent blinker.
- *Environment-independent hiding*, when $L_{\mathcal{H}}$ is defined by an I -independent blinker.

Note that temporal hiding is a special case of both environment- and system-independent hiding, thus hiding only depends on the number of rounds since the beginning of the interaction. Also, while in system-independent hiding the system has no influence on hiding, it may avoid hiding in environment-independent hiding, yet this may conflict with satisfying the specification.

A natural question is whether blinking synthesis becomes easier in the restricted cases. Since the blinkers used in the lower bounds in the proofs of Theorem 3.3 and Theorem 3.4 are both fixed and temporal (indeed, in the first, the hiding language contains all words, and in the

second it contains all words of even lengths), the specification (and hence also the general) complexity of blinking synthesis does not become easier. Hence, we have the following:

Theorem 3.7. *The specification complexities of LTL and DBW blinking synthesis with temporal hiding are 2EXPTIME-complete and EXPTIME-complete, respectively.*

We continue to the blinker complexity and show that there, environment-independent (and hence, also temporal) hiding reduces the complexity of blinking synthesis. In fact, the improved complexity applies to $(I, \mathcal{H})$ -independent blinkers, which generalize environment-independent visibility. Note that the blinker used in the lower bound proof in Theorem 3.6 is indeed not $(I, \mathcal{H})$ -independent.

In the full version, we prove the following theorem. Essentially, it follows from the fact that the universal automaton for the specification and blinker is deterministic in the blinker element, and thus the exponential blow-up in translating it to a DPW is only in the specification, which is fixed.

Theorem 3.8. *The blinker complexity of blinking synthesis with $(I, \mathcal{H})$ -independent blinkers is PTIME.*

3.4 Strong blinking synthesis

Defining blinking synthesis, we assume that the environment must hide input assignments after histories in $L_{\mathcal{H}}$. That is, $L_{\mathcal{H}}$ describes not a *possibility* that the input assignments are hidden but a *guarantee* that this is the case. In some settings, hiding after behaviors in $L_{\mathcal{H}}$ is not guaranteed and $L_{\mathcal{H}}$ only describes possible hiding, for example when sensors may or may not overcome some disruption. In this section we study *strong blinking synthesis*, which studies such settings. The strong semantics can be used in order to avoid systems that realize the specification by relaying on the assumptions that assignments are hidden in all prefixes in $L_{\mathcal{H}}$. As Example 3.1 demonstrates, while such a policy results in correct implementations, we typically prefer systems that do not rely on the environment violating assumptions [5].

Example 3.1. Let $I = \{x\}$, $O = \{y\}$, and $L_{\mathcal{H}} = \Sigma^*$. Consider the specification $\varphi = \text{G}y$ and the blinking transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ appearing in Figure 2. Both transducers blinkingly realize $\langle \varphi, L_{\mathcal{H}} \rangle$. While $\mathcal{T}_1$ does it in a straightforward manner, correctness of $\mathcal{T}_2$ is based on the fact that the system has to satisfy φ only when the environment hides the input assignments along the whole computation. Indeed, if the environment makes the value of x visible, the transducer $\mathcal{T}_2$ assigns False to y and φ is violated. $\square$

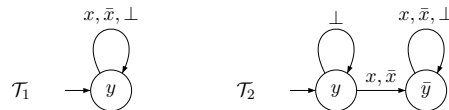


Figure 2: The transducers $\mathcal{T}_1$ and $\mathcal{T}_2$.

Below we define the strong semantics formally. For a word $w \in \Sigma^\omega$, let $\text{shide}_{L_{\mathcal{H}}}(w)$ be the set of all words obtained from w by changing some of the assignments to input signals after prefixes in $L_{\mathcal{H}}$ to $\perp$; Thus $\text{shide}_{L_{\mathcal{H}}}(w) = \bigcup_{L \subseteq L_{\mathcal{H}}} \text{hide}_L(w)$. Accordingly, for the reverse operator, let $\text{sfill}_{L_{\mathcal{H}}}(w) = \bigcup_{L \subseteq L_{\mathcal{H}}} \text{fill}_L(w)$. Note that for all $w \in \Sigma$ and $w' \in \Sigma_{\mathcal{H}}$, we have that $w' \in \text{sfill}_{L_{\mathcal{H}}}(w)$ iff $w \in \text{shide}_{L_{\mathcal{H}}}(w')$. For a specification $L_\varphi \subseteq \Sigma^\omega$ and a hiding language $L_{\mathcal{H}} \subseteq \Sigma^\omega$, we say that

a blinker transducer $\mathcal{T}$ strongly blinkingly realizes $\langle L_\varphi, L_{\mathcal{H}} \rangle$ if $\text{sfill}_{L_{\mathcal{H}}}(L(\mathcal{T})) \subseteq L_\varphi$. Then, the pair $\langle L_\varphi, L_{\mathcal{H}} \rangle$ is strongly blinkingly realizable iff there is a blinking transducer that strongly blinkingly realizes $\langle L_\varphi, L_{\mathcal{H}} \rangle$.

While knowing the values of input assignments may help a blinking transducer $\mathcal{T}$ to realize the specification, the fact $\mathcal{T}$ has to satisfy L_φ with all hidings that $L_{\mathcal{H}}$ over-approximates, in particular with $L_{\mathcal{H}}$ itself, makes strong blinking synthesis more difficult than blinking synthesis. Intuitively, it prevents the system from using $\perp$ inputs in order to learn the hidden assignments. Formally, we have the following.

Theorem 3.9. *Consider a specification $L_\varphi \subseteq \Sigma^\omega$ and a hiding language $L_{\mathcal{H}} \subseteq \Sigma^\omega$. If $\langle L_\varphi, L_{\mathcal{H}} \rangle$ is strongly blinkingly realizable, then it is blinkingly realizable. On the other hand, possibly $\langle L_\varphi, L_{\mathcal{H}} \rangle$ is blinkingly realizable but not strongly blinkingly realizable.*

Proof. First, if a blinking transducer $\mathcal{T}$ strongly blinkingly realizes $\langle \varphi, L_{\mathcal{H}} \rangle$, then, by definition, it blinkingly realizes $\langle \varphi, L_{\mathcal{H}} \rangle$. For the failure of the other direction, consider the pair $\langle \varphi_1, L_{\mathcal{H}} \rangle$ described in Example 2.2. Recall that the blinking transducer in Figure 1 blinkingly realizes it. On the other hand, $\langle \varphi_1, L_{\mathcal{H}} \rangle$ is not strongly blinkingly realizable. To see this, consider a word $w_I = \perp \cdot x^\omega$, and a blinking transducer $\mathcal{T}$. Let $\mathcal{T}(w_I) = o_1 \cdot o_2 \cdots$. Note that both $w_1 = (\bar{x} \cdot x^\omega) \oplus \mathcal{T}(w_I)$ and $w_2 = x^\omega \oplus \mathcal{T}(w_I)$ are in $\text{sfill}_{L_{\mathcal{H}}}(w_I \oplus \mathcal{T}(w_I))$. Now, if $o_2 = y$, then $w_1 \in \text{sfill}_{L_{\mathcal{H}}}(L(\mathcal{T})) \cap L(\neg\varphi_1)$, and if $o_2 = \bar{y}$, then $w_2 \in \text{sfill}_{L_{\mathcal{H}}}(L(\mathcal{T})) \cap L(\neg\varphi_1)$. In both cases, $\text{sfill}_{L_{\mathcal{H}}}(L(\mathcal{T})) \not\subseteq \varphi_1$, and we are done. $\square$

Solving strong blinking synthesis is similar to solving blinking synthesis, adjusting the defining of $\text{blink}(\varphi, L_{\mathcal{H}})$ to its strong variant. In the full version, we describe the proofs in detail. Here, we outline the idea. For $L_\varphi \subseteq \Sigma^\omega$ and $L_{\mathcal{H}} \subseteq \Sigma^*$, we define $\text{sblink}(\varphi, L_{\mathcal{H}}) = \{w \in \Sigma_{\mathcal{H}}^\omega : \text{sfill}_{L_{\mathcal{H}}}(w) \subseteq L_\varphi\}$ and prove a strong analogue of Theorem 3.1, namely that $\langle L_\varphi, L_{\mathcal{H}} \rangle$ is strongly blinkingly realizable iff $\text{sblink}(L_\varphi, L_{\mathcal{H}})$ is realizable. The construction in Lemma 3.2 can now be adjusted to generate, given a specification UCW $\mathcal{U}$ and a blinker $\mathcal{B}$, a UCW for $\text{sblink}(\mathcal{U}, \mathcal{B})$. The construction is similar to the one described for $\text{blink}(\mathcal{U}, \mathcal{B})$ except that now viewing input assignments in states in $\mathcal{H}$ does not lead to q_{acc} and requires $\mathcal{U}_{\text{sblink}}^{\mathcal{B}}$ to send copies in the same way it proceeds in states in $\mathcal{V}$. The construction enables us to lift all the upper bounds from Section 3 to the strong setting. As we show in the full version, the fact the blinkers used in the lower-bound proofs are environment-independent enables us to lift the lower bounds too.

Remark 3.1. It is possible to combine the usual and strong semantics. Formally, we can refine $L_{\mathcal{H}}$ to $L_{\mathcal{H}}^G$ and $L_{\mathcal{H}}^P$, where hiding is *guaranteed* after histories in $L_{\mathcal{H}}^G$ and is *possible* after histories in $L_{\mathcal{H}}^P$. A blinker in this setting partitions the set of states into three sets $\mathcal{V}$, $\mathcal{H}^G$, and $\mathcal{H}^P$. Our algorithms can be easily adjusted to this setting. In particular, the transition function of $\mathcal{U}_{\text{blink}}^{\mathcal{B}}$ now has the condition “ $q \in \mathcal{V} \cup \mathcal{H}^P$ and $i \neq \perp$ ” in the first case, and “ $q \in \mathcal{H}^P \cup \mathcal{H}^G$ and $i = \perp$ ” in the second case. Thus, all our results apply also to this richer setting. $\square$

4 Synthesizing a Visibility Language

So far, we considered settings in which the hiding language $L_{\mathcal{H}}$ is given. In this section we study two settings in which $L_{\mathcal{H}}$ is not given, and the designer has to generate, given a specification φ , both $L_{\mathcal{H}}$ and a blinking transducer that realizes $\langle \varphi, L_{\mathcal{H}} \rangle$. Clearly, the smaller $L_{\mathcal{H}}$ is, the easier it is to design a system that realizes φ . We consider two incentives for the designer to increase $L_{\mathcal{H}}$. The first concerns *resources*: the system has a budget with which it can “buy” visibility in a given blinker. The second concerns *privacy*: in addition to a specification, the

input to the synthesis problem includes *secrets*, and the synthesized blinking transducer should limit observability so that the truth values of the secrets remain unknown.

4.1 Blinking synthesis with a bounded visibility budget

In this setting we assume the input to the synthesis problem includes, rather than a blinker, a *cost-transducer*, which specifies the cost of enabling visibility after different histories. Formally, a cost-transducer $\mathcal{C} = \langle \Sigma, Q, q_0, \delta, \text{cost} \rangle$ is similar to a DFW, except that the acceptance condition is replaced by a function $\text{cost} : Q \rightarrow \mathbb{Z}$ mapping each state to the cost of enabling visibility after histories that reach it. Note that $\text{cost}(q)$ may be negative, indicating that hiding visibility in q requires resources.

For a cost transducer $\mathcal{C} = \langle \Sigma, Q, q_0, \delta, \text{cost} \rangle$ and a budget $b \in \mathbb{N}$, we say that a blinker $\mathcal{B}$ *respects* $\langle \mathcal{C}, b \rangle$ if there is a set $\mathcal{V} \subseteq Q$ such that $\sum_{q \in \mathcal{V}} \text{cost}(q) \leq b$, and $\mathcal{B} = \langle \Sigma, Q, q_0, \delta, \mathcal{V}, Q \setminus \mathcal{V} \rangle$. That is, $\mathcal{B}$ has the structure of $\mathcal{C}$, and the cost of the $\mathcal{V}$ -states in $\mathcal{B}$ is at most b . For a specification $L_\varphi \subseteq \Sigma^\omega$, a cost transducer $\mathcal{C}$, and a budget $b \in \mathbb{N}$, we say that $\langle L_\varphi, \mathcal{C} \rangle$ is *blinkingly realizable with budget b* if there is a blinker $\mathcal{B}$ that respects $\langle \mathcal{C}, b \rangle$, and $\langle L_\varphi, \mathcal{B} \rangle$ is blinkingly realizable. In the problem of *blinking synthesis with a bounded budget*, we get a specification L_φ , a cost transducer $\mathcal{C}$, and a budget $b \in \mathbb{N}$, and we have to return a blinking transducer that blinkingly realize $\langle L_\varphi, \mathcal{B} \rangle$, for a blinker $\mathcal{B}$ that respects $\langle \mathcal{C}, b \rangle$, or decide that no such transducer exists.

Since going over all candidate blinkers involves an exponential factor in the already exponential complexity of blinking synthesis, it is not hard to see that blinking synthesis with a bounded budget inherits the complexity of blinking synthesis (see proof in the full version).

Theorem 4.1. *The problem of blinking synthesis with a bounded budget is 2EXPTIME-complete for specifications in LTL, and is EXPTIME-complete for fixed specifications and for specifications in DPWs and fixed cost transducers.*

Since blinking synthesis is EXPTIME-hard for fixed specifications, revealing the increase in complexity that a bounded budget involves requires the definition of a class of specifications for which blinking synthesis can be solved in PTIME, whereas blinking synthesis with a bounded budget is NP-complete. Below we describe such a class.

Consider a blinker $\mathcal{B} = \langle 2^{I \cup O}, Q, q_0, \delta, \mathcal{V}, \mathcal{H} \rangle$ and a blinking transducer $\mathcal{T} = \langle \Sigma_{I, \mathcal{H}}, O, S, s_0, \mu, \tau \rangle$. Let $\mathcal{B}_\mathcal{T} = \langle \Sigma, (Q \times S) \cup \{q_{\text{sink}}\}, \langle s_0, q_0 \rangle, \delta', \mathcal{V} \times S, (\mathcal{H} \times S) \cup \{q_{\text{sink}}\} \rangle$ be the restriction of $\mathcal{B}$ to words generated by $\mathcal{T}$. Formally, for all states $\langle s, q \rangle$ of $\mathcal{B}_\mathcal{T}$ and letters $\langle i, o \rangle \in \Sigma$, we have that

$$\delta'(\langle q, s \rangle, \langle i, o \rangle) = \begin{cases} \langle \delta(q, \langle i, o \rangle), \mu(q, i) \rangle & \text{if } q \in \mathcal{V} \text{ and } o = \tau(\mu(q, i)) \\ \langle \delta(q, \langle i, o \rangle), \mu(q, \perp) \rangle & \text{if } q \in \mathcal{H} \text{ and } o = \tau(\mu(q, \perp)) \\ q_{\text{sink}} & \text{otherwise} \end{cases}$$

That is, $\mathcal{B}_\mathcal{T}$ agrees with $\mathcal{B}$ on the classification to $\mathcal{H}/\mathcal{V}$ of all all prefixes of words in $\text{fill}_\mathcal{B}(L(\mathcal{T}))$, and reaches q_{sink} with all words not generated by $\mathcal{T}$.

A specification φ is $(I/\mathcal{H})$ -independent if for every blinker $\mathcal{B}$ and blinking transducer $\mathcal{T}$ that blinkingly realizes $\langle \varphi, \mathcal{B} \rangle$, we have that $\mathcal{B}_\mathcal{T}$ is $(I/(\mathcal{H} \times S))$ -independent. Intuitively, the property requires that whenever φ can be realized with some information about the input being hidden, the behavior of the blinker cannot depend on this hidden information. For example, consider the specification $\varphi = G(x \leftrightarrow y)$, with $x \in I$ and $y \in O$. If the transducer reaches a hidden state, it must produce the same output for both possible values of x , and hence cannot satisfy $G(x \leftrightarrow y)$ for all fillings. Thus, every transducer realizing φ must avoid hidden states along every computation. Consequently, the only reachable states of $\mathcal{B}_\mathcal{T}$ are visible states (and q_{sink}), and the independence requirement holds vacuously. In contrast, consider $\varphi = Gy$. This

specification can be realized without observing the input, and hence a realizing transducer may reach hidden states. If the blinker is input-dependent at such a state, the resulting blinker $\mathcal{B}_{\mathcal{T}}$ need not be $(I/(\mathcal{H} \times S))$ -independent. Thus, Gy is not $(I/\mathcal{H})$ -independent.

The important feature of $(I/\mathcal{H})$ -independent specifications is that the UCW $\mathcal{U}_{\text{blink}}^{\mathcal{B}}$ constructed for them in Theorem 3.2 is deterministic in its blinker component. Hence, the synthesis procedure is exponential in the specification component, but only polynomial in the blinker component. Formally, we have the following (see missing details in the full version).

Theorem 4.2. *Blinking realizability of fixed $(I/\mathcal{H})$ -independent specifications can be solved in PTIME, yet blinking realizability with bounded budget for them is NP-complete.*

Proof. Let L_{φ} be an $(I/\mathcal{H})$ -independent specification, $\mathcal{B}$ be a blinker, and $\mathcal{T}$ be a blinking transducer that blinkingly realizes $\langle \varphi, \mathcal{B} \rangle$. As we prove in the full version, for every word $w \in L(\mathcal{T})$, the runs of $\mathcal{B}$ on words in $\text{fill}_{\mathcal{B}}(w)$ coincide, which leads to a PTIME algorithm for blinking realizability of fixed $(I/\mathcal{H})$ -independent specifications.

We continue to blinking realizability with a bounded budget. Let L_{φ} be an $(I/\mathcal{H})$ -independent specification, $\mathcal{C}$ be a cost transducer, and b a budget. For membership in NP, we can guess a blinker that respects $\mathcal{C}$ and b , and, by Theorem 3.8, check that $\langle L_{\varphi}, \mathcal{B} \rangle$ is blinkingly realizable in polynomial time. For the lower bound, we describe a reduction from the vertex-cover problem. The idea of the reduction is as follows (see full details in the full version). Consider a graph $G = \langle V, E \rangle$. In each round in the interaction, the environment provides an edge $e \in E$ and the system should respond with a vertex of e . The state space of the cost transducer is V , and it moves to the vertex suggested by the system. In order to respond correctly, the system must stay in $\mathcal{V}$ -states, and it can choose such $k \geq 1$ states and still respond correctly to all possible edges iff there is a vertex cover of size k . $\square$

4.2 Blinking synthesis that maintains privacy

In *blinking synthesis with privacy*, we are given, in addition to the specification language $\varphi \subseteq (2^{I \cup O})^{\omega}$, also a list of *secrets* $\psi_1, \dots, \psi_k \subseteq (2^{I \cup O})^{\omega}$, which defines behaviors that we want to hide from the system. That is, in the one hand, the system needs to increase visibility in order to satisfy its specification, and on the other hand, it should reduce visibility in order not to be able to valuate the truth value of the secrets.

Given φ and $\psi_1, \dots, \psi_k$, the goal is to generate both a blinking transducer and $\mathcal{T}$ and a blinker $\mathcal{B}$, such that the following hold:

- (Correctness) $\mathcal{T}$ blinkingly realizes $\langle \varphi, \mathcal{B} \rangle$.
- (Privacy) For every computation $w \in L(\mathcal{T})$ and every $1 \leq i \leq k$, there are two computations $w_1, w_2 \in \text{fill}_{\mathcal{B}}(w)$ such that $w_1 \in L(\psi_i)$ and $w_2 \notin L(\psi_i)$.

Note that the second condition means that the language $L(\mathcal{B}_{\mathcal{H}})$ hides the secrets from $\mathcal{T}$.

Blinking synthesis can be viewed as a special case of *synthesis with privacy* [17]. There, rather than generating $\mathcal{B}$, the synthesis algorithm has to direct the environment in each round which signals to hide. Technically, this is done by a transducer that in addition to assigning truth values to the output signals, has a mapping $\iota : S \rightarrow 2^I$ that labels each state by the subset of input signals whose values should be made visible in the next round. In the case of blinking synthesis all input signals are hidden together, so the mapping μ can be replaced by a simpler mapping $\mathbf{b} : S \rightarrow \{\mathcal{H}, \mathcal{V}\}$, indicating the rounds in which all input signals are hidden.

The blinker $\mathcal{B}$ is embodied in the blinking transducer $\mathcal{T}$. Formally, a *proactive blinking (I/O) -transducer* $\mathcal{T} = \langle \Sigma_{I,H}, O, S, s_0, \eta, \tau, \mathbf{b} \rangle$, augments an (I/O) -blinking transducer by a

blinking function $\mathbf{b} : S \rightarrow \{\mathcal{H}, \mathcal{V}\}$. When $\mathcal{T}$ is in state s , it generates the assignment $\tau(s)$ to the output signals and directs the system as to whether the values of all input signals should be hidden (when $\mathbf{b}(s) = \mathcal{H}$) or visible (when $\mathbf{b}(s) = \mathcal{V}$) in the next round. The blinker induced by $\mathcal{T}$ is then $\mathcal{B} = \langle 2^{I \cup O}, S, s_0, \delta, \mathcal{V}, \mathcal{H} \rangle$, where for every $s \in S$ and $\langle i, o \rangle \in 2^{I \cup O}$, we have that $\eta'(s, i \cup o) = \eta(s, i)$ if $o = \tau(\eta(s, i))$, and $\eta'(s, i \cup o)$ is arbitrary otherwise.

Since a proactive blinking transducer is a special case of a *noisy transducer* [17], the upper bound from [17] apply to our case. Since it agrees with that of LTL traditional synthesis, the lower bound applies too, and we can conclude with the following.

Theorem 4.3. *LTL blinking synthesis with privacy is 2EXPTIME-complete.*

We note that extending the tighter analysis described in Section 3.2 to a setting with privacy involves several challenges. First, in the parameterized analysis, the secrets replace the blinker. Second, even when the secrets are given by deterministic automata, the need to guess input assignments that witness different evaluations for the secrets has the effect of nondeterminism. Together with the effect of universality that hiding has on the specification, we end up with an alternating (rather than universal) automaton that is analogous to $\mathcal{U}_{\text{blink}}^{\mathcal{B}}$. Finally, a new and interesting issue is the size of the generated blinker, and its dependency on both the specification and the secrets.

References

- [1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 25th Int. Colloq. on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1989.
- [2] 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.
- [3] Ali Asadi, Krishnendu Chatterjee, David Lurie, and Raimundo Saona. Revealing pomdps: Qualitative and quantitative analysis for parity objectives. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 36146–36154, 2026.
- [4] R. Bloem, K. Chatterjee, and B. Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking.*, pages 921–962. Springer, 2018.
- [5] R. Bloem, R. Ehlers, and R. Könighofer. Cooperative reactive synthesis. In *13th Int. Symp. on Automated Technology for Verification and Analysis*, pages 394–410, 2015.
- [6] 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.
- [7] 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.
- [8] A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.
- [9] E.M. Clarke, O. Grumberg, D. Kroening, D. Peled, and H. Veith. *Model Checking, Second Edition*. MIT Press, 2018.
- [10] D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.
- [11] E. Filiot, N. Jin, and J-F. Raskin. An antichain algorithm for LTL realizability. In *Proc. 21st Int. Conf. on Computer Aided Verification*, volume 5643, pages 263–277, 2009.

- [12] 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.
- [13] J. Goubault. Rigid e-unifiability is dextime-complete. In *Proc. 9th ACM/IEEE Symp. on Logic in Computer Science*, pages 498–506, 1994.
- [14] D. Harel and A. Pnueli. On the development of reactive systems. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 477–498. Springer, 1985.
- [15] T.A. Henzinger, S.C. Krishnan, O. Kupferman, and F.Y.C. Mang. Synthesis of uninitialized systems. In *Proc. 29th Int. Colloq. on Automata, Languages, and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 644–656. Springer, 2002.
- [16] O. Kupferman. Games with full, longitudinal, and transverse observability. In *14th International Conference on Reachability Problems*, volume 12448 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2020.
- [17] O. Kupferman and O. Leshkowitz. On repetition languages. In *45th Int. Symp. on Mathematical Foundations of Computer Science*, Leibniz International Proceedings in Informatics (LIPIcs), 2020.
- [18] O. Kupferman and N. Shenwald. Perspective games with notifications. In *Proc. 40th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 182 of *LIPIcs*, pages 51:1–51:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- [19] O. Kupferman and N. Shenwald. Perspective multi-player games. In *Proc. 36th ACM/IEEE Symp. on Logic in Computer Science*, pages 1–13, 2021.
- [20] O. Kupferman and G. Vardi. Perspective games. *ACM Trans. Comput. Log.*, 25(1):4:1–4:26, 2024.
- [21] O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.
- [22] D. Liberzon. *Switching in Systems and Control*. Birkhauser, 2003.
- [23] M. Margaliot. Stability analysis of switched systems using variational principles: an introduction. *Automatica*, 42(12):2059–2077, 2006.
- [24] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
- [25] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.
- [26] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. 16th Int. Colloq. on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989.
- [27] B. Puchala. Asynchronous omega-regular games with partial information. In *35th Int. Symp. on Mathematical Foundations of Computer Science*, pages 592–603. Springer, 2010.
- [28] J.H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and Systems Science*, 29:274–301, 1984.
- [29] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.
- [30] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.