Synthesis with Guided Environments*

Orna Kupferman and Ofer Leshkowitz
The Hebrew University of Jerusalem, Israel

Abstract

In the synthesis problem, we are given a specification, and we automatically generate a system that satisfies the specification in all environments. We introduce and study *synthesis with guided environments* (SGE, for short), where the system may harness the knowledge and computational power of the environment during the interaction. The underlying idea in SGE is that in many settings, in particular when the system serves or directs the environment, it is of the environment's interest that the specification is satisfied, and it would follow the guidance of the system. Thus, while the environment is still hostile, in the sense that the system should satisfy the specification no matter how the environment assigns values to the input signals, in SGE the system assigns values to some output signals and guides the environment via *programs* how to assign values to other output signals. A key issue is that these assignments may depend on input signals that are hidden from the system but are known to the environment, using programs like "copy the value of the hidden input signal x to the output signal y." SGE is thus particularly useful in settings where the system has partial visibility.

We solve the problem of SGE, show its superiority with respect to traditional synthesis, and study theoretical aspects of SGE, like the complexity (memory and domain) of programs used by the system, as well as the connection of SGE to synthesis of (possibly distributed) systems with partial visibility.

1 Introduction

Synthesis is the automated construction of a system from its specification [6]. Given a linear temporal logic (LTL) formula φ over sets I and O of input and output signals, the goal is to return a *transducer* that *realizes* φ. At each moment in time, the transducer reads a truth assignment, generated by the environment, to the signals in I, and generates a truth assignment to the signals in O. This process continues indefinitely, generating an infinite computation in

*A preliminary version was published in TACAS 2025. Supported in part by the Israel Science Foundation, Grant 2357/19, and the European Research Council, Advanced Grant ADVANSYNT.

$(2^{I \cup O})^\omega$. The transducer realizes φ if its interactions with all environments generate computations that satisfy φ [27].

The fact the system has to satisfy its specification in all environments has led to a characterization of the environment as *hostile*. In particular, in the game that corresponds to synthesis, the objective of the environment is to violate the specification φ. In real-life applications, the satisfaction of the specification is often also in the environment's interest. In particular, in cases where the system serves or guides the environment, we can expect the environment to follow guidance from the system. We introduce and study *synthesis with guided environments* (SGE, for short), where the system may harness the knowledge and computational power of the environment during the interaction, guiding it how to assign values to some of the output signals.

Specifically, in SGE, the set O of output signals is partitioned into sets C and G of *controlled* and *guided* signals. Then, a system that is synthesized by SGE assigns values to the signals in C and guides the environment how to assign values to the signals in G. Clearly, not all output signals may be guided to the environment. For example, physical actions of the system, like closing a gate or raising a robot arm, cannot be performed by the environment. In addition, it may be the case that the system cannot trust the environment to follow instructions for some of the output signals. As argued above, however, in many cases we can expect the environment to follow the system's guidance, and require the system to satisfy the specification only when the environment follows its guidance. Indeed, when we go to the cinema and end up seeing a bad movie, we cannot blame a recommendation system that guided us not to choose this movie. The recommendation system is bad (violates its specification, in our synthesis story) only if a user that follows its guidance is disappointed.[1]

One advantage of SGE is that it enables a decomposition of the satisfaction task between the system and the environment. We will get back to this point after we describe the setting in more detail. The main advantage of SGE, however, has to do with *partial visibility*, namely the setting where the set I of input signals is partitioned into sets V and H of *visible* and *hidden* signals, and the systems views only the signals in V. Partial visibility makes synthesis much harder, as the hidden signals still appear in the specification, yet the behavior of the system should be independent of their truth values.

Synthesis with partial visibility has been the subject of extensive research. One line of work studies the technical challenges in solving the problem [29, 23, 9]. Essentially, in a setting with full visibility, the interaction between the system and the environment induces a single computation and hence a single

[1] Readers who are still concerned about harnessing the environment towards the satisfaction of the specification, please note that the environment being hostile highlights the fact that the system has to satisfy the specification for *all* input sequences. This is still the case also in SGE: While we expect the environment to obey instructions received from the system, these instructions do not limit the input sequences that the environment may generate. Thus, there are no assumptions on the environment or collaboration between the system and the environment in the senses studied in [7, 4]: the setting is as in traditional synthesis, only with assignments to some output signals being replaced by programs that the environment is expected to follow.

run of a deterministic automaton for the specification. Partial visibility forces the algorithm to maintain subsets of states of the automaton. Indeed, now the interaction induces several computations, obtained by the different possible assignments to the hidden signals. This makes synthesis with partial visibility exponentially harder than synthesis with full visibility for specifications given by automata. When the specification is given by an LTL formula, this exponential price is dominated by the doubly-exponential translation of LTL formulas to deterministic automata, thus LTL synthesis with partial visibility is 2EXPTIME-complete, and is not harder than LTL synthesis with full visibility [30].

A second line of work studies different settings in which partial visibility is present. This includes, for example, *distributed systems* [28, 24, 32], systems with controlled *sensing* [11, 2] or with assumptions about visibility [16], and systems that maintain *privacy* [36, 21]. Finally, researchers have studied alternative forms of partial visibility (e.g., *perspective games*, where visibility of all signals is restricted in segments of the interaction), as well as partial visibility in richer settings (e.g., *multi-agent* [3, 5, 18] or *stochastic* [19, 10] systems) or in problems that are strongly related to synthesis (e.g., control [20], planning [12], and rational synthesis [8, 15]).

To the best of our knowledge, in all settings in which synthesis with partial visibility has been studied so far, there is no attempt to make use of the fact that the assignments to the hidden signals are known to the environment. In SGE, the instructions that the system sends to the environment may refer to the values of the hidden signals. Thus, the outcome of SGE is a *transducer with a guided environment* (TGE): a transducer whose transitions depend only on the assignments to the signals in V, it assigns values to the signals in C, and instructs the environment how to assign values to the signals in G using programs that may refer to the values of the signals in H.

Consider, for example, a system in a medical clinic that directs patients "if you come for a vaccine, go to the first floor; if you come to the pharmacy, go to the second floor". Such a system is as correct as a system that asks customers for the purpose of their visit and then outputs the floor to which they should go. Clearly, if the customers prefer not to reveal the purpose of their visit, we can direct them only with a system of this type. This is exactly what SGE does: it replaces assignments that depend on hidden signals or are complicated to compute with instructions to the environment. As another example, consider a smart-home controller that manages various smart devices within a home by getting inputs from devices like thermostats and security cameras, and generating outputs to devices like lighting systems or smart locks. When it is desirable to hide from the controller information like sleep patterns or number of occupants, we can do that by limiting its input, and guiding the user about the activation of some output devices, for example with instructions like, "if you expect guests, unlock the backyard gate". In Examples 2.1 and 2.2, we describe more elaborated examples, in particular of a server that directs users who want to upload data to a cloud. The users may hide from the server the sensitivity level of their data, and we can expect them to follow instructions that the server

issues, for example instructions to use storage of high security only when the data they upload is sensitive.

We study several aspects of SGE. We start by examining the memory used by the environment. Clearly, this memory can be used to reduce the state space of the TGE. To see this, note that in an extreme setting in which the TGE can guide all output signals, it can simply instruct the environment to execute a transducer that realizes the specification. Beyond a trade-off between the size of the TGE and the memory of the environment, we discuss how the size of the memory depends on the sets of visible and guided signals. For example, we show that, surprisingly, having more guided signals does not require more memory, yet having fewer guided signals may require more memory.

We then describe an automata-based solution for SGE. The main challenge in SGE is as follows. Consider a system that views only input signals in V. Its interactions with environments that agree on the signals in V generate the same response. In synthesis with partial visibility, this is handled by *universal tree automata* that run on trees with directions in 2^V, thus trees whose branches correspond to the interaction from the point of view of the system [23]. In the setting of SGEs, the interactions with different environments that agree on the signals in V still generate the same response, but this response now involves programs that guide the environment on how to assign values to signals in G based on the values assigned to the signals in H. As a result, the computations induced by these interactions may differ (not only on H but also on G). This differences between SGE and traditional synthesis with partial visibility is our main technical challenge. We show that we can still reduce SGE to the nonemptiness problem of universal co-Büchi tree automata, proving that SGE for LTL specifications is 2EXPTIME-complete. In more detail, given an LTL formula φ and a bound k on the memory that the environment may use, the constructed automaton is of size exponential in $|\varphi|$ and linear in k, making SGE doubly-exponential in $|\varphi|$ and exponential in k. Finally, we prove a doubly-exponential upper bound on the size of the memory needed by the environment, leading to an overall triply-exponential upper bound for the SGE problem with unbounded environment memory.

We continue and study the domain of programs that TGEs use. Recall that these programs instruct the environment how to update its memory and assign values to the guided signals, given a current memory state and the current assignment to the hidden signals. Thus, for a memory state space M, each program is of the form $p : M \times 2^H \to M \times 2^G$. We study ways to reduce the domain 2^H, which is the most dominant factor. The reduction depends on the specification φ we wish to synthesize. We argue that the SGE algorithm can restrict attention to *tight* programs: ones whose domain is a set of predicates over H obtained by simplifying propositional sub-formulas of φ. Further simplification is achieved by exploiting the fact that programs are called after an assignment to the signals in $V \cup C$ has been fixed, and exploiting dependencies among all signals.

Finally, we compare our solution with one that views a TGE as two distributed processes that are executed together in a pipeline architecture: the

TGE itself, and a transducer with state space M that implements the instructions of the TGE to the environment. We argue that the approach we take is preferable and can lead to a quadratic saving in their joint state spaces, similar to the saving obtained by defining a regular language as the intersection of two automata. Generating programs that manage the environment's memory efficiently is another technical challenge in SGE.

We conclude with directions for future research. Beyond extensions of the many settings in which synthesis has been studied to a setting with guided environments, we discuss two directions that are more related to "the guided paradigm" itself: settings with *dynamic* hiding and guidance of signals, thus when H and G are not fixed throughout the interaction; and *bounded SGE*, where, as in traditional synthesis [33, 22], beyond a bound on the memory used by the environment, there are bounds on the size of the state space of the TGE and possibly also on the size of the state space of its environment.

2 Preliminaries

We describe on-going behaviors of reactive systems using the linear temporal logic LTL [26]. We consider systems that interact via sets I and O of input and output signals, respectively. Formulas of LTL are defined over $I \cup O$ 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 all computations that satisfy φ.

The *length* of an LTL formula φ, denoted $|\varphi|$, is the number of nodes in the generating tree of φ. Note that $|\varphi|$ bounds the number of sub-formulas of φ.

We model reactive systems that interact with their environments by finite-state transducers. A *transducer* is a tuple $\mathcal{T} = \langle I, O, S, s_0, \delta, \tau \rangle$, 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, $\delta : S \times 2^I \to S$ is a transition function, and $\tau : S \times 2^I \to 2^O$ is a function that labels each transition by an assignment to the output signals. Given an infinite sequence $w_I = i_1 \cdot i_2 \cdots \in (2^I)^\omega$ of assignments to input signals, $\mathcal{T}$ generates an infinite sequence $w_O = o_1 \cdot o_2 \cdots \in (2^O)^\omega$ of assignments to output signals. Formally, a *run* of $\mathcal{T}$ on w_I is an infinite sequence of states $s_0 \cdot s_1 \cdot s_2 \cdots$, where for all $j \geq 1$, we have that $s_j = \delta(s_{j-1}, i_j)$. Then, the sequence w_O is obtained from the assignments along the transitions that the run traverses. Thus for all $j \geq 1$, we have that $o_j = \tau(s_{j-1}, i_j)$. We define the *computation* of $\mathcal{T}$ on w_I to be the word $\mathcal{T}(w_I) = (i_1 \cup o_1) \cdot (i_2 \cup o_2) \cdots \in (2^{I \cup O})^\omega$.

For a specification language $L_\varphi \subseteq (2^{I \cup O})^\omega$, we say that $\mathcal{T}$ (I, O)-*realizes* L_φ if for every input sequence $w_I \in (2^I)^\omega$, we have that $\mathcal{T}(w_I) \in L_\varphi$. In the *synthesis* problem, we are given a specification language L_φ and a partition of the signals to I and O, and we have to return a transducer that (I, O)-realizes L_φ (or determine that L_φ is not realizable). The language L_φ is typically given by an LTL formula φ. We then talk about realizability or synthesis of φ (rather than L_φ).

In synthesis with *partial visibility*, we seek a system that satisfies a given specification in all environments even when it cannot observe the assignments to some of the input signals. Formally, the set of input signals is partitioned into *visible* and *hidden* signals, thus $I = V \cup H$. The specification φ is still over $V \cup H \cup O$, yet the behavior of the transducer that models the system is independent of H. Formally, $\mathcal{T} = \langle V, H, O, S, s_0, \delta, \tau \rangle$, where now $\delta : S \times 2^V \to S$ and $\tau : S \times 2^V \to 2^O$. Given an infinite sequence $w_I = (v_1 \cup h_1) \cdot (v_2 \cup h_2) \cdots \in (2^{V \cup H})^\omega$, the run and computation of $\mathcal{T}$ on w_I is defined as in the case of full visibility, except that now, for all $j \geq 1$, we have that $s_j = \delta(s_{j-1}, v_j)$ and $o_j = \tau(s_{j-1}, v_j)$.

We can now define a *transducer with a guided environment* (TGE for short). TGEs extend traditional transducers by instructing the environment how to manage its guided signals. A TGE may be executed in a setting with partial visibility, thus $I = V \cup H$. It uses the fact that the assignments to the signals in H are known to the environment, and it guides the assignment to some of the output signals to the environment. As discussed in Section 1, in practice not all output signals can be guided. Formally, the set of output signals is partitioned into *controlled* and *guided* signals, thus $O = C \cup G$. In each transition, the TGE assigns values to the signals in C and instructs the environment how to assign values to the signals in G. The environment may have a finite memory, in which case the transducer also instructs the environment how to update the memory in each transition. The instructions that the transducer generates are represented by *programs*, defined below.

Consider a finite set M of memories, and sets $H \subseteq I$ and $G \subseteq O$ of input and output signals. Let $\mathcal{P}_{M,H,G} = (M \times 2^G)^{M \times 2^H}$ denote the set of *propositional programs* that update the memory state and assign values to signals in G, given a memory in M and an assignment to the signals in H. Note that each member of $\mathcal{P}_{M,H,G}$ is of the form $p : M \times 2^H \to M \times 2^G$. For a program $p \in \mathcal{P}_{M,H,G}$, let $p_M : M \times 2^H \to M$ and $p_G : M \times 2^H \to 2^G$ be the projections of p onto M and G respectively. Thus, $p(m, h) = \langle p_M(m, h), p_G(m, h) \rangle$ for all $\langle m, h \rangle \in M \times 2^H$. In Section 5 we discuss ways to restrict the set of programs that a TGE may suggest to its environment without affecting the outcome of the synthesis procedure.

Now, a *TGE* is $\mathcal{T} = \langle V, H, C, G, S, s_0, \delta, M, m_0, \tau \rangle$, where V and H are sets of visible and hidden input signals, C and G are sets of controlled and guided output signals, S is a finite set of states, $s_0 \in S$ is an initial state, $\delta : S \times 2^V \to S$ is a transition function, M is a set of memories that the environment may use, $m_0 \in M$ is an initial memory, and $\tau : S \times 2^V \to 2^C \times \mathcal{P}_{M,H,G}$ labels each transition by an assignment to the controlled output signals and a program. Note that δ and τ are independent of the signals in H, and that τ assigns values to the signals in C and instructs the environment how to use the signals in H in order to assign values to the signals in G. The latter reflects the fact that the environment does view the signals in H, and constitute the main advantage of TGEs over standard transducers.

Given an infinite sequence $w_I = (v_1 \cup h_1) \cdot (v_2 \cup h_2) \cdots \in (2^{V \cup H})^\omega$, the run of $\mathcal{T}$ on w_I is obtained by applying δ on the restriction of w_I to V. Thus, $r = s_0 \cdot$

$s_1 \cdot s_2 \cdots$, where for all $j \geq 1$, we have that $s_j = \delta(s_{j-1}, v_j)$. The interaction of $\mathcal{T}$ with the environment generates an infinite sequence $w_C = c_1 \cdot c_2 \cdot c_3 \cdots \in (2^C)^\omega$ of assignments to the controlled signals, an infinite sequence $w_M = m_0 \cdot m_1 \cdot m_2 \cdots \in M^\omega$ of memories, and an infinite sequence $w_P = p_1 \cdot p_2 \cdot p_3 \cdots \in (\mathcal{P}_{M,H,G})^\omega$ of programs, which in turn generates an infinite sequence $w_G = d_1 \cdot d_2 \cdot d_3 \cdots \in (2^G)^\omega$ of assignments to the guided signals. Formally, for all $j \geq 1$, we have that $\langle c_j, p_j \rangle = \tau(s_{j-1}, v_j)$ and $\langle m_j, d_j \rangle = p_j(m_{j-1}, h_j)$. The computation of $\mathcal{T}$ on w_I is then $\mathcal{T}(w_I) = (v_1 \cup h_1 \cup c_1 \cup d_1) \cdot (v_2 \cup h_2 \cup c_2 \cup d_2) \cdots \in (2^{V \cup H \cup C \cup G})^\omega$.

Note that while the domain of the programs in $\mathcal{P}_{M,H,G}$ is $M \times 2^H$, programs are chosen in $\mathcal{T}$ along transitions that depend on 2^V. Thus, effectively, assignments made by programs depend on signals in both H and V.

For a specification language $L_\varphi \subseteq (2^{I \cup O})^\omega$, partitions $I = V \cup H$ and $O = C \cup G$, and a bound $k \geq 1$ on the memory that the environment may use, we say that a TGE $\mathcal{T}$, *(V, H, C, G)-realizes L_φ with memory k*, if $\mathcal{T} = \langle V, H, C, G, S, s_0, \delta, M, m_0, \tau \rangle$, with $|M| = k$, and $\mathcal{T}(w_I) \in L_\varphi$, for every input sequence $w_I \in (2^I)^\omega$. Then, in the *synthesis with guided environment* problem (SGE, for short), given such a specification $L_\varphi \subseteq (2^{I \cup O})^\omega$, a bound $k \geq 1$ and partitions $I = V \cup H$ and $O = C \cup G$, we should construct a TGE with memory k that (V, H, C, G)-realizes L_φ (or determine that no such TGE exists).

Let us consider the special case of TGEs that operate in an environment with no memory, thus $M = \{m\}$, for a single memory state m. Consider first the setting where $\mathcal{T}$ has full visibility, thus $V = I$ and $H = \emptyset$. Then, programs are of the form $p : \{m\} \times 2^\emptyset \to \{m\} \times 2^G$, and so each program is a fixed assignment to the signals in G. Hence, TGEs that have full visibility and operate in an environment with no memory coincide with traditional transducers. Consider now a setting with partial visibility, thus $V \neq I$. Now, programs are of the form $p : \{m\} \times 2^H \to \{m\} \times 2^G$, allowing the TGE to guide the environment in a way that depends on signals in H.

In Examples 2.1 and 2.2 below, we demonstrate how collaboration from the environment can lead to the realizability of specifications that are otherwise non-realizable.

Example 2.1 [TGEs for simple specifications] Let $I = \{i\}$, $O = \{o\}$, and consider the simple specification $\varphi_1 = \mathbf{G}(i \leftrightarrow o)$. Clearly, φ_1 is realizable in a setting with full visibility, yet is not realizable in a setting with partial visibility and $H = \{i\}$. A TGE can realize φ_1 even in the setting with partial visibility. Indeed, even when the environment has no memory, it can guide the assignment to o to the environment and instruct it to copy the value of i into o. Formally, φ_1 is realizable by the $\mathcal{T}_1 = \langle \emptyset, \{i\}, \emptyset, \{o\}, \{s\}, \{s\}, \delta, \{m\}, m, \tau \rangle$, with $\delta(s, \emptyset) = s$ and $\tau(s, \emptyset) = \langle \emptyset, p \rangle$, where the program $p \in \mathcal{P}_{\{m\},\{i\},\{o\}}$ is the command $o := i$.

Consider now the specification $\varphi_2 = \mathbf{G}(i \leftrightarrow \mathbf{X}o)$. Here, a TGE in a setting in which i is not visible needs an environment with at least one register, inducing a memory of size 2. Then, the TGE can instract the environment to store the value of i in the register, and use the stored value when assigning a value to the signal o in the next round. Formally, φ_2 is realizable by the TGE $\mathcal{T}_2 = \langle \emptyset, \{i\}, \emptyset, \{o\}, \{s\}, \{s\}, \delta, \{m_0, m_1\}, m_0, \tau \rangle$,with $\delta(s, \emptyset) = s$ and $\tau(s, \emptyset) = \langle \emptyset, p \rangle$,

where $p \in \mathcal{P}_{\{m_0,m_1\},\{i\},\{o\}}$ instructs the environment to move to m_0 when $i = \mathtt{F}$ and to m_1 when $i = \mathtt{T}$, and to assigns $\mathtt{F}$ to o when it is in m_0 and $\mathtt{T}$ when it is in m_1. □

Example 2.2 [A TGE implementing a server] Consider a server that directs users who want to upload data to a cloud. The set of input signals is $I = \{req, sens\}$, where *req* holds when a user requests to upload data, and *sens* holds when the data is sensitive. The set of output signals is $O = \{open, high\}$, where *open* holds when the cloud is open for uploading, and *high* holds when the storage space is of high security. Users pay more for storage of high security. Therefore, in addition to guaranteeing that all requests are eventually responded by *open*, we want the server to direct the users to use storage of high security only when the data they upload is sensitive. In addition, the cloud cannot stay always open to uploads. Formally, we want to synthesize a transducer that realizes the conjunction φ of the following LTL formulas.

- $\varphi_1 = \mathbf{G}((req \wedge sens) \rightarrow ((\neg open)\mathbf{U}(open \wedge high)))$.
- $\varphi_2 = \mathbf{G}((req \wedge \neg sens) \rightarrow ((\neg open)\mathbf{U}((open \wedge \neg high) \vee (req \wedge sens))))$.
- $\varphi_3 = \mathbf{GF}\neg open$.

A server that has full visibility of I can realize φ. To see this, note that a server can open the cloud for uploading whenever a request arrives, storing the data in a storage of high security iff it is sensitive. Then, in order to satisfy φ_3, the server should delay the response to successive requests, but this does not prevent it from satisfying φ_1 and φ_2.

Users may prefer not to share with the server information about the sensitivity of their data. In current settings of synthesis with partial visibility, this renders φ to be unrealizable. Indeed, when the sensitivity of the data is hidden from the scheduler, thus when $V = \{req\}$ and $H = \{sens\}$, the behavior of the server is independent of *sens*, making it impossible to assign values to *high* in a way that satisfies φ in all environments.

While the output signal *open* controls access to the cloud, the output signal *high* only directs the user which type of storage to use, and it is of the user's interest to store her data in a storage of an appropriate security level. Accordingly, the environment can be guided with the assignment to *high*. Thus, $C = \{open\}$ and $G = \{high\}$.

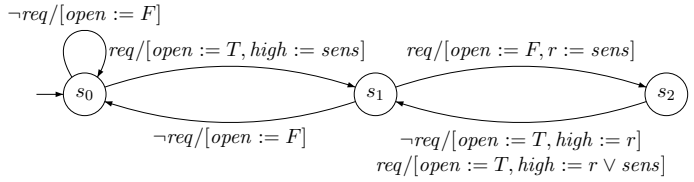

Figure 1: A transducer with a guided environment.

In Figure 1, we present a TGE that implements such a guidance and realizes φ. Each transition of the TGE is labeled by a guarded command of the form $v/[c, p]$, where $v \in 2^V$ is an assignment to the visible input signals (or a predicate describing several assignments), $c \in 2^C$ is an assignment to the controlled output signals, and p is a *program* that instructs the environment how to assign values to the guided signals and how to update its *memory*.[2] The environment uses a memory with one register r. Accordingly, the transitions of the TGE are guarded by the truth value of *req*, they include an assignment to *open*, and then a program that instructs the environment how to assign a value to *high* and to the register r. The key advantage of TGEs is that these instructions may depend on the hidden input signals. Indeed, the environment does know their values. For example, when the TGE moves from s_1 to s_2, it instructs the environment to assign to r the value of *sens*. Then, when the TGE moves from s_2 to s_1, it instructs the environment to use the value stored in r (as well as the current value of *sens*) when it assigns a value to *high*. □

3 On the Memory Used by the Environment

In this section we examine different aspects of the memory used by the environment. We discuss a trade-off between the size of the memory and the size of the TGE, and how the two depend on the partitions $I = V \cup H$ and $O = C \cup G$ of the input and output signals.

Example 3.1 In order to better understand the need for memory and the technical challenges around it, consider the tree appearing in Figure 2.

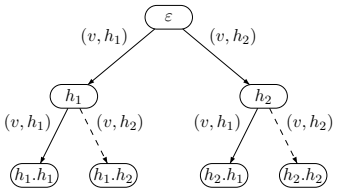

Figure 2: Two rounds of interaction with a TGE $\mathcal{T}$.

The tree describes two rounds of an interaction between a TGE $\mathcal{T}$ and its environment. In both rounds, $\mathcal{T}$ views the assignment $v \in 2^V$. Consider two assignments $h_1, h_2 \in 2^H$. The tree presents the four possible extensions of the interaction by assignments in $\{h_1, h_2\}$. Since the signals in H are hidden, $\mathcal{T}$ cannot distinguish among the different branches of the tree. In particular, it has to suggest to the environment the same program in all the transitions

[2] For clarity, in the TGE in Figure 1, the assignment to *high* is missing in transitions with *open* = `F`, where *high* can be assigned arbitrarily, and the assignment to the register r is missing in transitions that are taken when there are no pending requests, in which case r can be assigned arbitrarily.

from the root to its successors, and the same program in all the transitions from these successors to the leaves. While these programs may depend on H, a program with no memory would issue the same assignment to the signals in G along the two dashed transitions $\langle h_1, h_1.h_2\rangle$ and $\langle h_2, h_2.h_2\rangle$. Indeed, in both transitions, the environment assigns $h_2 \in 2^H$ to the hidden signals. Using memory, $\mathcal{T}$ can instruct the environment to maintain information about the history of the computation, and thus distinguish between computations with the same visible component and the same current assignment to the hidden signals. For example, by moving to memory state m_1 with h_1 and to memory state m_2 with h_2, a program may instruct the environment to assign different values along the dashed lines. □

We first observe that in settings in which all output signals can be guided, the TGE can instruct the environment to simulate a transducer that realizes the specification. Also, in settings with full visibility, the advantage of guided environments is only computational, thus specifications do not become realizable, yet may be realizable by smaller systems. Formally, we have the following.

Theorem 3.2 *Consider an LTL specification φ over $I \cup O$.*

1. *If φ is realizable by a transducer with n states, then φ is $(\emptyset, I, \emptyset, O)$-realizable by a one-state TGE with memory n.*
2. *For every partition $I = V \cup H$ and $O = C \cup G$, if φ is (V, H, C, G)-realizable by a TGE with n states and memory k, then φ is (I, O)-realizable (in a setting with full visibility) by a transducer with $n \cdot k$ states. In particular, if φ is $(\emptyset, I, \emptyset, O)$-realizable by a one-state TGE with memory k, then φ is (I, O)-realizable (in a setting with full visibility) by a transducer with k states.*

Proof: We start with the first claim. Given a transducer $\mathcal{T} = \langle I, O, S, s_0, \delta, \tau\rangle$, we define a program $p \in \mathcal{P}_{S,I,O}$ that instructs the environment to simulate $\mathcal{T}$. Formally, for every $s \in S$ and $i \in 2^I$, we have that $p(s, i) = \langle \delta(s, i), \tau(s, i)\rangle$. Consider the TGE $\mathcal{T}' = \langle \emptyset, I, \emptyset, O, \{s\}, s, \delta', S, s_0, \tau'\rangle$, with $\delta'(s, \emptyset) = s$ and $\tau'(s, \emptyset) = \langle \emptyset, p\rangle$. It is easy to see that for every $w_I \in (2^I)^\omega$, we have that $\mathcal{T}(w_I) = \mathcal{T}'(w_I)$. In particular, if $\mathcal{T}$ realizes φ, then so does $\mathcal{T}'$.

We continue to the second claim. Note that the special case where $V = C = \emptyset$ is the "only if" direction of the first claim. Given a TGE $\mathcal{T} = \langle V, H, C, G, S, s_0, \delta, M, m_0, \tau\rangle$, consider the transducer $\mathcal{T}' = \langle H \cup V, C \cup G, S \times M, \langle s_0, m_0\rangle, \delta', \tau'\rangle$, where for all $\langle s, m\rangle \in S \times M$ and $i \in 2^{H \cup V}$ with $\tau(s, i \cap H) = \langle c, p\rangle$, we have that $\delta'(\langle s, m\rangle, i) = \langle \delta(s, i \cap V), p_M(i \cap H)\rangle$ and $\tau'(\langle s, m\rangle, i) = c \cup p_G(i \cap H)$. It is easy to see that for every $w_I \in (2^I)^\omega$, we have that $\mathcal{T}(w_I) = \mathcal{T}'(w_I)$. In particular, if $\mathcal{T}$ realizes φ, then so does $\mathcal{T}'$. □

Note that Theorem 3.2 also implies that an optimal balance between controlled and guided signals in a setting with full visibility can achieve at most a quadratic saving in the combined states space of the TGE and its memory.

This is similar to the quadratic saving in defining a regular language by the intersection of two automata. Indeed, handling of two independent specifications can be partitioned between the TGE and the environment. Formally, we have the following.

Theorem 3.3 *For all $k \geq 1$, there exists a specification φ_k over $I = \{i_1, i_2\}$ and $O = \{o_1, o_2\}$, such that φ_k is $(\{i_1\}, \{i_2\}, \{o_1\}, \{o_2\})$-realizable by a TGE with k states and memory k, yet a transducer that (I, O)-realizes φ_k needs at least k^2 states.*

Proof: For a proposition p, let $\mathbf{F}^1 p = \mathbf{F}p$, and for $j \geq 1$, let $\mathbf{F}^{j+1} p = \mathbf{F}(p \wedge \mathbf{X}(\mathbf{F}^j p))$. Thus, $\mathbf{F}^j p$ holds in a computation π iff p holds in at least j positions in π.

We define $\varphi_k = \varphi_k^1 \wedge \varphi_k^2$, where for $l \in \{1, 2\}$, we have that $\varphi_k^l = (\mathbf{F}^k i_l) \leftrightarrow \mathbf{F}o_l$. That is, φ_k^l is over $\{i_l, o_l\}$, and it holds in a computation π if o_l is always `F` in π, unless there are k positions in π in which i_l is `T`, in which case o_l has to be `T` in at least one position.

We prove that an (I, O)-transducer that realizes φ_k needs k^2 states, yet there is a TGE that $(\{i_1\}, \{i_2\}, \{o_1\}, \{o_2\})$-realizes φ_k with state space and memory set both of size k.

It is easy to see that φ_k is $(\{i_l\}, \{o_l\})$-realizable by a transducer $\mathcal{T}_l^k$ with k states $\{s_0, \ldots, s_{k-1}\}$. The state space serves like a counter. Transitions from state s_j, for $0 \leq j < k-1$, assign `F` to o_l, looping when i_1 is `F` and moving to s_{j+1} when i_1 is `T`. From state s_{k-1}, the transducer always loops, copying the value of i_l to o_l.

A TGE that $(\{i_1\}, \{i_2\}, \{o_1\}, \{o_2\})$-realizes φ_k can have the state space of $\mathcal{T}_1^k$, and label its transitions by a program p that instructs the environment to simulates $\mathcal{T}_2^k$, with memory that corresponds to the state space of $\mathcal{T}_2^k$. The details are similar to the simulation described in the proof of Theorem 3.2 (1).

On the other hand, a transducer that (I, O)-realizes φ_k needs to maintain both a counter for i_1 and a counter for i_2, and so it needs at least k^2 states. The formal proof is similar to the proof that an automaton for $F^k i_1 \wedge F^k i_2$ needs k^2 states. Essentially, a transducer with fewer states would reach the same state s after reading two input sequences that differ in the number of occurrences of i_1 or i_2, and would then err on the output of at least one continuation of these sequences. □

Unsurprisingly, larger memory of the environment enables TGEs to synthesize more specifications. Formally, we have the following.

Theorem 3.4 *For every $k \geq 1$, the LTL specification $\varphi_k = \mathbf{G}(i \leftrightarrow \mathbf{X}^k o)$ is $(\emptyset, \{i\}, \emptyset, \{o\})$-realizable by a TGE with memory 2^k, and is not with memory 2^{k-1}.*

Proof: We first describe a TGE with memory that is composed of k registers $\{r_1, r_2, \ldots, r_k\}$, that $(\emptyset, \{i\}, \emptyset, \{o\})$-realize φ_k. The TGE has a single state in which it instructs the environment to assign to o the value stored in r_k, and to

shift the stored values to the right, thus with r_1 getting the current value of i and r_{j+1} getting the value stored in r_j, for $1 \leq j < k$. This guarantees that the value assigned to o agrees with the value that i was assigned with k rounds earlier, and so φ_k is satisfied in all environments.

We continue and prove the lower bound. I.e., that a TGE with less than 2^{k-1} memory states cannot $(\emptyset, \{i\}, \emptyset, \{o\})$-realize φ_k. Consider a TGE $\mathcal{T}$ that $(\emptyset, \{i\}, \emptyset, \{o\})$-realizes φ with memory set M. As $\mathcal{T}$ has no visibility, it generates the same sequence $\langle \emptyset, p^1 \rangle, \langle \emptyset, p^2 \rangle, \langle \emptyset, p^3 \rangle, \ldots \in (2^{\{\emptyset\}} \times \mathcal{P}_{M,\{i\},\{o\}})^\omega$, for all input sequences. Consider the function $f : (2^{\{i\}})^* \to M$ that maps a finite input sequence $x \in (2^{\{i\}})^*$ to the memory in M that is reached after the environment generated x. Thus, $f(\varepsilon) = m_0$, and for every $x \in (2^{\{i\}})^*$ and $a \in 2^{\{i\}}$, we have $f(x \cdot a) = p_M^{|x|+1}(f(x), a)$.

We prove that f is injective on $(2^{\{i\}})^k$. That is, we prove that for every two input sequences $x, y \in (2^{\{i\}})^k$, if $x \neq y$, then $f(x) \neq f(y)$. It will then follow that $|M| \geq 2^k$, as required.

Assume by way of contradiction that there are two input sequences $x = x_1 \cdot x_2 \cdots x_{k-1} \in (2^{\{i\}})^{k-1}$ and $y = y_1 \cdot y_2 \cdots y_{k-1} \in (2^{\{i\}})^{k-1}$, such that $x \neq y$, yet $f(x) = f(y) = m$. Since $x \neq y$, there is some position $t \leq k-1$ such that $x_t \neq y_t$.

Consider the infinite input sequences $x' = x \cdot \emptyset^\omega$ and $y' = y \cdot \emptyset^\omega$. By our assumption, $\mathcal{T}$ reaches the same memory state m after the first $k-1$ rounds of its interaction with environments that generate x' and y'. Since x' and y' agree on their suffix after these rounds, we have that $\mathcal{T}(x')$ and $\mathcal{T}(y')$ agree on the assignment to o on all positions after k. In particular, $\mathcal{T}(x')$ and $\mathcal{T}(y')$ agree on the assignment to o in position $k+t$, contradicting the fact that i holds only in one of the assignments x_t and y_t. Thus, $\mathcal{T}$ does not $(\emptyset, \{i\}, \emptyset, \{o\})$-realizes φ, and we have reached a contradiction. □

We turn to consider how changes in the partition of the input signals to visible and hidden signals, and the output signals to controlled and guided ones, may affect the required memory. In Theorem 3.5, we show that increasing visibility or decreasing control, does not require more states or memory. On the other hand, in Theorem 3.6, we show that increasing control by the system may require the environment to use more memory. Thus, surprisingly, guiding more signals does not require more memory, yet guiding fewer signals may require more memory.

Theorem 3.5 *Consider an LTL specification φ that is (V, H, C, G)-realizable by a TGE with n states and memory k.*

1. *For every $I' \subseteq H$, we have that φ is $(V \cup I', H \setminus I', C, G)$-realizable by a TGE with n states and memory k.*
2. *For every $O' \subseteq C$, we have that φ is $(V, H, C \setminus O', G \cup O')$-realizable by a TGE with n states and memory k.*

Proof: Let $\mathcal{T} = \langle V, H, C, G, S, s_0, \delta, M, m_0, \tau \rangle$ be a TGE that (V, H, C, G)-realizes φ. We start with the first claim. Given $I' \subseteq H$, consider the TGE $\mathcal{T}' = \langle V \cup I', H \setminus I', C, G, S, s_0, \delta', M, m_0, \tau' \rangle$, where for every $s \in S$ and $u \in 2^{V \cup I'}$, we have that $\delta'(s, u) = \delta(s, u \cap V)$, and if $\tau(s, u \cap V) = \langle c, p \rangle$, then $\tau'(s, u) = \langle c, p' \rangle$, where for all $g \in 2^{H \setminus I'}$, we have that $p'(g) = p(g \cup (u \cap I'))$. It is easy to see that $\mathcal{T}'$ has the same size and memory as $\mathcal{T}$, and that for every $w_I \in (2^I)^\omega$, we have that $\mathcal{T}(w_I) = \mathcal{T}'(w_I)$. In particular, if $\mathcal{T}$ realizes φ, then so does $\mathcal{T}'$.

We continue to the second claim. Given $O' \subseteq C$, consider the TGE $\mathcal{T}' = \langle V, H, C \setminus O', G \cup O', S, s_0, \delta, M, m_0, \tau' \rangle$, where for every $s \in S$ and $v \in 2^V$ with $\tau(s, v) = \langle c, p \rangle$, we have that $\tau'(s, v) = \langle c \setminus O', p' \rangle$, where for every $h \in 2^H$ with $p(h) = \langle m, d \rangle$, we have $p'(h) = \langle m, d \cup (c \cap O') \rangle$. It is easy to see that $\mathcal{T}'$ has the same size and memory as $\mathcal{T}$, and that for every $w_I \in (2^I)^\omega$, we have that $\mathcal{T}(w_I) = \mathcal{T}'(w_I)$. In particular, if $\mathcal{T}$ realizes φ, then so does $\mathcal{T}'$. □

Theorem 3.6 *There is an LTL specification φ over $\{i, o_1, o_2\}$ such that the following hold:*

1. *φ is $(\emptyset, \{i\}, \emptyset, \{o_1, o_2\})$-realizable by a TGE with memory 1.*
2. *φ is $(\emptyset, \{i\}, \{o_1\}, \{o_2\})$-realizable by a TGE with memory 2.*
3. *φ is not $(\emptyset, \{i\}, \{o_1\}, \{o_2\})$-realizable by a TGE with memory 1.*

Proof: We define $\varphi = \varphi_1 \vee \varphi_2$, with $\varphi_1 = \mathbf{G}(i \leftrightarrow o_1)$ and $\varphi_2 = \mathbf{G}(i \leftrightarrow \mathbf{X}o_2)$. A TGE with no memory that $(\emptyset, \{i\}, \emptyset, \{o_1, o_2\})$-realizes φ, instructs the environment to always copy i into o_1, and thus realizes φ_1. A TGE that does not guide o_1, cannot satisfy φ_1, as i is hidden, and should thus instruct the environment in a way that causes the realization of φ_2. This, however, must involve a register that maintain the previous value of i. □

4 Solving the SGE Problem

Recall that in the SGE problem, we are given an LTL specification φ over $I \cup O$, partitions $I = V \cup H$ and $O = C \cup G$, and a bound $k \geq 1$, and we have to return a TGE that (V, H, C, G)-realizes φ with memory k. The main challenge in solving the SGE problem is that it adds to the difficulties of synthesis with partial visibility the fact that the system's interactions with environments that agree on the visible signals may generate different computations. Technically, the system should still behave in the same way on input sequences that agree on the visible signals. Thus, the interaction should generate the same assignments to the controlled output signals and the same sequence of programs. However, these programs may generate different computations, as they also depend on the hidden signals.

Our solution uses *alternating tree automata*, and we start by defining them.

4.1 Words, trees, and automata

An *automaton* on infinite words is $\mathcal{A} = \langle \Sigma, Q, q_0, \eta, \alpha \rangle$, where Σ is an alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\eta : Q \times \Sigma \to 2^Q$ is a transition function, and α is an acceptance condition to be defined below. If $|\eta(q, \sigma)| = 1$ for every state $q \in Q$ and letter $\sigma \in \Sigma$, then $\mathcal{A}$ is *deterministic*.

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 $r = r_0 \cdot r_1 \cdot r_2 \cdots \in Q^\omega$, such that $r_0 = q_0$, and for all $i \geq 0$, we have that $r_{i+1} \in \eta(r_i, \sigma_{i+1})$. The acceptance condition α defines a subset of Q^ω, indicating which runs are *accepting*. We consider here the *Büchi*, *co-Büchi*, and *parity* acceptance conditions. All conditions refer to the set $inf(r) \subseteq Q$ of states that r traverses infinitely often. Formally, $inf(r) = \{q \in Q : q = r_i \text{ for infinitely many } i\text{'s}\}$. In a Büchi automaton, the acceptance condition is $\alpha \subseteq Q$ and a run r is accepting if $inf(r) \cap \alpha \neq \emptyset$. Thus, r visits α infinitely often. Dually, in a co-Büchi automaton, a run r is accepting if $inf(r) \cap \alpha = \emptyset$. Thus, r visits α only finitely often. Finally, in a parity automaton $\alpha : Q \to \{1, ..., k\}$ maps states to ranks, and a run r is accepting if the maximal rank of a state in $inf(r)$ is even. Formally, $\max_{q \in inf(r)}\{\alpha(q)\}$ is even. A run that is not accepting is *rejecting*.

Note that when $\mathcal{A}$ is not deterministic, it has several runs on a word. If $\mathcal{A}$ is a *nondeterministic* automaton, then a word w is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. If $\mathcal{A}$ is a *universal* automaton, then a word w is accepted by $\mathcal{A}$ if all the runs of $\mathcal{A}$ on w are accepting. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts.

Given a set Υ of directions, the *full Υ-tree* is the set $T = \Upsilon^*$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. An *edge* in T is a pair $\langle x, x \cdot a \rangle$, for $x \in \Upsilon^*$ and $a \in \Upsilon$. The node $x \cdot a$ is called a *successor* of x. A *path* π of T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, there exists a unique $a \in \Upsilon$ such that $x \cdot a \in \pi$. We associate an infinite path π with the infinite word in Υ^ω obtained by concatenating the directions taken along π.

Given an alphabet Σ, a *Σ-labeled Υ-tree* is a pair $\langle T, \ell \rangle$, where $T = \Upsilon^*$ and ℓ labels each edge of T by a letter in Σ. That is, $\ell(x, x \cdot a) \in \Sigma$ for all $x \in T$ and $a \in \Upsilon$. Note that an infinite word in Σ^ω can be viewed as a Σ-labeled $\{a\}$-tree.

A *universal tree automaton* over Σ-labeled Υ-trees is $\mathcal{A} = \langle \Sigma, \Upsilon, Q, q_0, \eta, \alpha \rangle$, where Σ, Q, q_0, and α are as in automata over words, Υ is the set of directions, and $\eta : Q \times \Upsilon \times \Sigma \to 2^Q$ is a transition function.

Intuitively, $\mathcal{A}$ runs on an input Σ-labeled Υ-tree $\langle T, \ell \rangle$ as follows. The run starts with a single copy of $\mathcal{A}$ in state q_0 that has to accept the subtree of $\langle T, \ell \rangle$ with root ε. When a copy of $\mathcal{A}$ in state q has to accept the subtree of $\langle T, \ell \rangle$ with root x, it goes over all the directions in Υ. For each direction $a \in \Upsilon$, it proceeds according to the transition function $\eta(q, a, \sigma)$, where σ is the letter written on the edge $\langle x, x \cdot a \rangle$ of $\langle T, \ell \rangle$. The copy splits into $|\eta(q, a, \sigma)|$ copies: for each $q' \in \eta(q, a, \sigma)$, a copy in state q' is created, and it has to accept the subtree with root $x \cdot a$.

Formally, a *run* of $\mathcal{A}$ over a Σ-labeled Υ-tree $\langle T, \ell \rangle$, is a tree with directions in $\Upsilon \times Q$ and nodes labeled by pairs in $T \times Q$ that describe how the different

copies of $\mathcal{A}$ proceed. Formally, a run is a pair $\langle T_r, r\rangle$, where $T_r \subseteq (\Upsilon \times Q)^*$ and $r : T_r \to T \times Q$ are defined as follows.

- $\varepsilon \in T_r$ and $r(\varepsilon) = \langle \varepsilon, q_0\rangle$. Thus, the run starts with a single copy of $\mathcal{A}$ that reads the root of $\langle T, \ell\rangle$ and is in state q_0.
- Consider a node $y \in T_r$ with $r(y) = \langle x, q\rangle$. Recall that y corresponds to a copy of $\mathcal{A}$ that reads the node x of $\langle T, \ell\rangle$ and is in state q. For a direction $a \in \Upsilon$ let $\sigma_a = \ell(x, x \cdot a)$. For every state $q' \in \eta(q, a, \sigma_a)$, we have that $y \cdot \langle a, q'\rangle \in T_r$ and $r(y \cdot \langle a, q'\rangle) = \langle x \cdot a, q'\rangle$. Thus, the run sends $|\eta(q, a, \sigma_a)|$ copies to the subtree with root $x \cdot a$, one for each states in $\eta(q, a, \sigma_a)$.

Acceptance is defined as in automata on infinite words, except that now we define, given a run $\langle T_r, r\rangle$ and an infinite path $\pi \subseteq T_r$, the set $inf(\pi) \subseteq Q$ as the set of states that are visited along π infinitely often, thus $q \in inf(\pi)$ if and only if there are infinitely many nodes $y \in \pi$ for which $r(y) \in T \times \{q\}$. We denote by $L(\mathcal{A})$ the set of all Σ-labeled Υ-trees that $\mathcal{A}$ accepts.

We denote the different classes of automata by three-letter acronyms in $\{\text{D,N,U}\} \times \{\text{B,C,P}\} \times \{\text{W,T}\}$. The first letter stands for the branching mode of the automaton (deterministic, nondeterministic, or universal); the second for the acceptance condition type (Büchi, co-Büchi, or parity); and the third indicates we consider automata on words or trees. For example, UCTs are universal co-Büchi tree automata.

4.2 An automata-based solution

Each TGE $\mathcal{T} = \langle V, H, C, G, S, s_0, \delta, M, m_0, \tau\rangle$ induces a $(2^C \times \mathcal{P}_{M,H,G})$-labeled 2^V-tree $\langle (2^V)^*, \ell\rangle$, obtained by simulating the interaction of $\mathcal{T}$ with all input sequences. Formally, let $\delta^* : (2^V)^* \to S$ be an extension of δ to finite sequence in $(2^V)^*$, starting from s_0. Thus, $\delta^*(w)$ is the state that $\mathcal{T}$ visits after reading $w \in (2^V)^*$. Formally, $\delta^*(\varepsilon) = s_0$, and for $w \in (2^V)^*$ and $v \in 2^V$, we have that $\delta^*(w \cdot v) = \delta(\delta^*(w), v)$. Then, the tree $\langle (2^V)^*, \ell\rangle$ is such that $\ell(w, w \cdot v) = \tau(\delta^*(w), v)$.

Given $w_V = v_1 \cdot v_2 \cdot v_3 \cdots \in (2^V)^\omega$, let $\ell(w_V) = \langle c_1, p_1\rangle \cdot \langle c_2, p_2\rangle \cdot \langle c_3, p_3\rangle \cdots \in (2^C \times \mathcal{P}_{M,H,G})^\omega$ be the sequence of labels along the path induced by w_V. For every $j \geq 1$, $\langle c_j, p_j\rangle = \ell(\langle v_1, v_2, \ldots v_{j-1}\rangle, \langle v_1, v_2, \ldots, v_j\rangle)$. It is convenient to think of $\ell(w_V)$ as the operation of $\mathcal{T}$ while interacting with an environment that generates an input sequence $(v_1 \cup h_1) \cdot (v_2 \cup h_2) \cdot (v_2 \cup h_2) \cdots$, for some $w_H = h_1 \cdot h_2 \cdot h_3 \cdots \in (2^H)^\omega$. $\mathcal{T}$ sees only V, knowing its assignments to C and the programs for the environment, but not the H assignments, and thus neither the memory state nor the G assignments.

The environment, which does know w_H, can complete $\ell(w_V)$ to a computation in $(2^{V \cup H \cup C \cup G})^\omega$. Indeed, given a current memory state $m \in M$ and assignment $h \in 2^H$ to the hidden signals, the environment can apply the last program sent $p \in \mathcal{P}_{M,H,G}$, $p(m, h) = \langle m', g\rangle$, and thus obtain the new memory state and assignment to the guided signals. Formally, for all $j \geq 1$, we have

that $\langle m_j, g_j \rangle = p_j(m_{j-1}, h_j)$. Then, the *operation* of $\ell(w_V)$ on w_H, denoted $\ell(w_V) \circ w_H$, is the sequence $(v_1 \cup h_1 \cup c_1 \cup g_1) \cdot (v_2 \cup h_2 \cup c_2 \cup g_2) \cdots \in (2^{V \cup H \cup C \cup G})^\omega$.

For an LTL specification φ, we say that a $(2^C \times \mathcal{P}_{M,H,G})$-labeled 2^V-tree $\langle (2^V)^*, \ell \rangle$ is *φ-good* if for all $w_V \in (2^V)^\omega$ and $w_H \in (2^H)^\omega$, we have that $\ell(w_V) \circ w_H$ satisfies φ. By definition, a TGE (V, H, C, G)-realizes φ iff its induced $(2^C \times \mathcal{P}_{M,H,G})$-labeled 2^V-tree is φ-good.

Theorem 4.1 *Given an LTL specification φ over $I \cup O$, partitions $I = V \cup H$ and $O = C \cup G$, and a set M of memories, we can construct a UCT $\mathcal{A}$ with $2^{|\varphi|} \cdot |M|$ states such that $\mathcal{A}_\varphi$ runs on $(2^C \times \mathcal{P}_{M,H,G})$-labeled 2^V-trees and accepts a labeled tree $\langle (2^V)^*, \ell \rangle$ iff $\langle (2^V)^*, \ell \rangle$ is φ-good.*

Proof: Given φ, let $\mathcal{A}_\varphi = \langle 2^{I \cup O}, Q, q_0, \eta, \alpha \rangle$ be a UCW over the alphabet $2^{I \cup O}$ that recognizes L_φ. We can construct $\mathcal{A}_\varphi$ by dualizing an NBW for $\neg\varphi$. By [35], the latter is of size exponential in $|\varphi|$, and thus, so is $\mathcal{A}_\varphi$.

We define $\mathcal{A} = \langle 2^C \times \mathcal{P}_{M,H,G}, 2^V, Q \times M, \langle q_0, m_0 \rangle, \eta', \alpha \times M \rangle$, where $m_0 \in M$ is arbitrary, and η' is defined for every $\langle q, m \rangle \in Q \times M$, $v \in 2^V$, and $\langle c, p \rangle \in 2^C \times \mathcal{P}_{M,H,G}$, as follows.

$$\eta'(\langle q, m \rangle, v, \langle c, p \rangle) = \bigcup_{h \in 2^H} \eta(q, v \cup h \cup c \cup p_G(m, h)) \times \{p_M(m, h)\}.$$

Thus, to direction $v \in 2^V$, the UCT $\mathcal{A}$ sends copies that correspond to all possible assignments in 2^H, where for every $h \in 2^H$, a copy is sent for each state in $\eta(q, v \cup h \cup c \cup p_G(m, h))$, all with the same memory $p_M(m, h)$. Accordingly, for all $2^C \times \mathcal{P}_{M,H,G}$-labeled 2^V-tree $\langle (2^V)^*, \ell \rangle$, there is a one to one correspondence between every infinite branch in the run tree $\langle T_r, r \rangle$ of $\mathcal{A}$ over $\langle (2^V)^*, \ell \rangle$, and an infinite run of the UCW $\mathcal{A}_\varphi$ over $\ell(w_V) \circ w_H$, for some $w_V \in (2^V)^\omega$ and $w_H \in (2^H)^\omega$. It follows that $\mathcal{A}$ accepts a tree $\langle (2^V)^*, \ell \rangle$ iff $\langle (2^V)^*, \ell \rangle$ is φ-good. □

The construction of $\mathcal{A}_\varphi$ allows us to conclude the following complexity result of the SGE problem.

Theorem 4.2 *The LTL SGE problem is 2EXPTIME-complete. Given an LTL formula φ over sets of signals V, H, C, and G, and a integer k, deciding whether φ is (V, H, C, G)-realizable by a TGE with memory k can be done in time doubly exponential in $|\varphi|$ and exponential in k.*

Proof: We start with the lower bound which follows immediately by the fact that traditional LTL synthesis is 2EXPTIME-complete [30]. The traditional synthesis of a specification φ above $I \cup O$ is cleary reduced to the synthesis of a TGE that $(I, \emptyset, O, \emptyset)$-realizes φ. Indeed, a TGE with $H = \emptyset$ and $G = \emptyset$ is simply a traditional transducer as its transmitted programs are redundant and do not affect the generated computation. Thus, since LTL synthesis is 2EXPTIME-hard, we conclude that so is the SGE problem.

We continue and prove the upper bound. Let M be a memory of size k, and let $\mathcal{A}$ be the UCT constructed for φ, V, H, C, G, and M in Theorem 4.1. As $\mathcal{A}$ accepts exactly all φ-good $(2^C \times \mathcal{P}_{M,H,G})$-labeled 2^V-trees, we can reduce SGE to the non-emptiness problem for $\mathcal{A}$, returning a witness when it is non-empty.

In [25] the authors solved the non-emptiness problem for UCTs by translating a UCT $\mathcal{A}$ into an NBT $\mathcal{A}'$ such that $L(\mathcal{A}') \subseteq L(\mathcal{A})$ and $L(\mathcal{A}') \neq \emptyset$ iff $L(\mathcal{A}) \neq \emptyset$. The translation goes from a UCT with n states to an NBT with $2^{O(n^2 \log n)}$ states.[3]

Consider an NBT with state space S that runs on Σ-labeled Υ-trees. Each nondeterministic transition of the NBT maps an assignment Σ^Υ (of labels along the branches that leave the current node) to a set of functions in S^Υ (describing possible labels by states of the successors of the current node in the run tree). Accordingly, non-emptiness of the NBT can be solved by deciding a Büchi game in which the OR-vertices are the states in S, and the AND-vertices are functions in S^Υ [17].

In our case, as $\mathcal{A}$ has $k \cdot 2^{|\varphi|}$ states, we have that S is of size $2^{O(k^2 \log k 2^{O(|\varphi|)})}$ and $|\Upsilon| = 2^{|V|}$. Hence, $|S^\Upsilon|$, which is the main factor in the size of the game, is $2^{O(k^2 \log k 2^{O(|\varphi|)} 2^{|V|})}$. Since $|V| \leq |\varphi|$ and since Büchi games can be solved in quadratic time [34], we end up with the required complexity. □

4.3 Bounding the size of M

As discussed in Example 3.1, programs that can only refer to the current assignment of the hidden signals may be too weak: in some specifications, the assignment to the guided output signals have to depend on the history of the interaction so far. The full history of the interaction is a finite word in $(2^{I \cup O})^*$, and so apriori, an unbounded memory is needed in order to remember all possible histories. A deterministic automaton D_φ for φ partitions the infinitely many histories in $(2^{I \cup O})^*$ into finitely many equivalence classes. Two histories $w, w' \in (2^{I \cup O})^*$ are in the same equivalence class if they reach the same state of D_φ, which implies that $w \cdot t \models \varphi$ iff $w' \cdot t \models \varphi$ for all $t \in (2^{I \cup O})^\omega$. In the case of traditional synthesis, we know that a transducer that realizes φ does not need more states than D_φ. Intuitively, if two histories of the interaction are in the same equivalence class, the transducer can behave the same way after processing them.

In the following theorem we prove that the same holds for the memory used by a TGE: if two histories of the interaction reach the same state of D_φ, there is no reason for them to reach different memories. Formally, we have the following.

Theorem 4.3 *Consider a specification φ, and let Q be the set of states of a DPW for φ. If there is a TGE that (V, H, C, G)-realizes φ with memory M, then there is also a TGE that (V, H, C, G)-realizes φ with memory Q.*

[3] The construction in [25] refers to automata with labels on the nodes rather than the edges, but it can be easily adjusted to edge-labeled trees.

Proof: Let $\mathcal{T} = \langle V, H, C, G, S, s_0, \delta, M, m_0, \tau \rangle$ be a TGE that (V, H, C, G)-realizes φ, and let $\mathcal{T}_{(V/C)} = \langle V, C, S, s_0, \delta, \tau_C \rangle$ be the (V, C)-transducer induced by $\mathcal{T}$. The labeling function $\tau_C : S \times 2^V \to 2^C$ is obtained by projecting τ on its 2^C component. Let $D_\varphi = \langle 2^{I \cup O}, Q, q_0, \eta, \alpha \rangle$ be a DPW for φ. We show that there exists a labeling function $\tau_G : S \times 2^V \to \mathcal{P}_{Q,H,G}$ such that $\mathcal{T}_Q = \langle V, H, C, G, S, s_0, \delta, Q, q_0, \tau' \rangle$, with $\tau'(s, v) = \langle \tau_C(s, v), \tau_G(s, v) \rangle$ is a TGE that (V, H, C, G)-realizes φ.

Consider the following two player game $\mathcal{G}_{\mathcal{T}, D_\varphi}$ played on top of the product of $\mathcal{T}_{(V/C)}$ with D_φ. The game is played between the environment, which proceeds in positions in $S \times Q$, and the system, which proceeds in positions in $S \times Q \times 2^I$. The game is played from the initial position $\langle s_0, q_0 \rangle$, which belongs to the environment. From a position $\langle s, q \rangle$, the environment chooses an assignment $i \in 2^I$ and moves to $\langle s, q, i \rangle$. From a position $\langle s, q, i \rangle$, the system chooses an assignment $g \in 2^G$ and moves to $\langle s', q' \rangle$, where $s' = \delta(s, i \cap V)$ and $q' = \eta(q, i \cup \tau_C(s, i \cap V) \cup g)$. The system wins the game if the Q component of the generated play satisfies α.

Note that the system wins iff there exists a strategy $f : (2^I)^* \to 2^G$ that generates for each $w_I \in (2^I)^\omega$, a word $f(w_I) \in (2^G)^\omega$ such that $w_I|_H \oplus \mathcal{T}_{(V/C)}(w_I|_V) \oplus f(w_I) \in L(D_\varphi)$. Indeed, the Q-component in the outcome of the game when the environment plays $w_I \in (2^I)^\omega$ is precisely the run of D_φ on the word $w = w_I \cup w_C \cup f(w_I)$, where w_C is the word generated by the (V/C)-transducer $\mathcal{T}_{(V/C)}$ on the input word $w_I|_V$, and, by definition $\mathcal{T}_{(V/C)}(w_I|_V) = w_I|_V \oplus w_C$. Moreover, by the memoryless determinacy of parity games, such a strategy f exists iff there exists a winning positional strategy $g : S \times Q \times 2^I \to 2^G$. Thus, instead of considering the entire history in $(2^I)^*$, the system has a winning strategy that only depends on the current position in $S \times Q$.

For all $s \in S$ and $v \in 2^V$, let $p_{s,v} : Q \times 2^H \to Q \times 2^G$ be defined by $p_{s,v}(q, h) = \langle q', d \rangle$, where $d = g(s, q, v \cup h)$ and $q' = \eta(q, v \cup h \cup \tau_C(s, v) \cup d)$. Finally, let $\tau_Q : S \times 2^V \to \mathcal{P}_{Q,H,G}$ be defined by $\tau_Q(s, v) = p_{s,v}$, and $\mathcal{T}_Q = \langle V, H, C, G, S, s_0, \delta, Q, q_0, \tau_Q \rangle$ be the TGE with memory Q that is obtained from $\mathcal{T}$ by replacing M, m_0, and τ with Q, q_0, and τ_Q, respectively.

It is not hard to see that for all $w_I \in (2^I)^\omega$, the computation $\mathcal{T}_Q(w_I)$ is exactly the outcome of the game when Sys plays with the winning strategy g. I.e., $\mathcal{T}_Q(w_I) = w_I|_H \oplus \mathcal{T}_{(V/C)}(w_I|_V) \oplus g(w_I)$. Hence, $\mathcal{T}_Q(w_I) \in L(D_\varphi)$, for all $w_I \in (2^I)^\omega$, as required. □

Theorem 4.4 *If there is a TGE that (V, H, C, G)-realizes an LTL specification φ, then there is also a TGE that (V, H, C, G)-realizes φ with memory doubly exponential in $|\varphi|$, and this bound is tight.*

Proof: The upper bound follows from Theorem 4.3 and the doubly-exponential translation of LTL formulas to DPWs [35, 31, 14]. The lower bound follows from the known doubly-exponential lower bound on the size of transducers for LTL formulas [30], applied when $I = H$ and $O = G$. □

While Theorem 4.4 is of theoretical interest, we find the current formulation of the SGE problem, which includes a bound on M, more appealing in practice: recall that SGE is doubly-exponential in $|\varphi|$ and exponential in $|M|$. As $|D_\varphi|$ is already doubly exponential in $|\varphi|$, solving SGE with no bound on M results in an algorithm that is triply-exponential in $|\varphi|$. Thus, it makes sense to let the user provide a bound on the memory.

5 Programs

Recall that TGEs generate in each transition a program $p : M \times 2^H \to M \times 2^G$ that instructs the environment how to update its memory and assign values to the guided output signals. In this section we discuss ways to represent programs efficiently, and, in the context of synthesis, restrict the set of programs that a TGE may suggest to its environment without affecting the outcome of the synthesis procedure. Note that the number of programs in $\mathcal{P}_{M,H,G}$ is $2^{(\log|M|+|G|)\cdot|M|\cdot 2^{|H|}}$. Our main goal is to reduce the domain 2^H, which is the most dominant factor.

Naturally, the reduction depends on the specification we wish to synthesize. For an LTL formula ψ, let $\mathit{prop}(\psi)$ be the set of maximal predicates over $I \cup O$ that are subformulas of ψ. Formally, $\mathit{prop}(\psi)$ is defined by an induction on the structure of ψ as follows.

- If ψ is a propositional assertions, then $\mathit{prop}(\psi) = \{\psi\}$.
- Otherwise, ψ is of the form $*\psi_1$ or $\psi_1 * \psi_2$ for some (possibly temporal) operator $*$, and $\mathit{prop}(\psi) = \mathit{prop}(\psi_1) \cup \mathit{prop}(\psi_2)$.

Note that the definition is sensitive to syntax. For example, the formulas $(i_1 \vee o) \wedge \mathbf{X} i_2$ and $(i_1 \wedge \mathbf{X} i_2) \vee (o \wedge \mathbf{X} i_2)$ are equivalent, but have different sets of maximal propositional assertions. Indeed, $\mathit{prop}((i_1 \vee o) \wedge \mathbf{X} i_2) = \{i_1 \vee o, i_2\}$, whereas $\mathit{prop}((i_1 \wedge \mathbf{X} i_2) \vee (o \wedge \mathbf{X} i_2)) = \{i_1, i_2, o\}$.

It is well known that the satisfaction of an LTL formula ψ in a computation $\pi \in (2^{I \cup O})^\omega$ depends only on the satisfaction of the formulas in $\mathit{prop}(\psi)$ along π: if two computations agree on $\mathit{prop}(\psi)$, then they also agree on ψ. Formally, for two assignments $\sigma, \sigma' \in 2^{I \cup O}$, and a set Θ of predicates over $I \cup O$, we say that σ and σ' *agree on* Θ, denoted $\sigma \approx_\Theta \sigma'$, if for all $\theta \in \Theta$, we have that $\sigma \models \theta$ iff $\sigma' \models \theta$. Then, two computations $\pi = \sigma_1 \cdot \sigma_2 \cdots$ and $\pi' = \sigma'_1 \cdot \sigma'_2 \cdots$ in $(2^{I \cup O})^\omega$ agree on Θ, denoted $\pi \approx_\Theta \pi'$, if for all $j \geq 1$, we have $\sigma_j \approx_\Theta \sigma'_j$.

Proposition 5.1 *Consider two computations $\pi, \pi' \in (2^{I \cup O})^\omega$. For every LTL formula ψ, if $\pi \approx_{\mathit{prop}(\psi)} \pi'$, then $\pi \models \psi$ iff $\pi' \models \psi$.*

As we shall formalize below, Proposition 5.1 enables us to restrict the set of programs so that only one computation from each equivalence class of the relation $\approx_{\mathit{prop}(\psi)}$ may be generated by the interaction of the system and the environment.

For an LTL formula ψ, let $cl_H(\psi)$ be the set of maximal subformulas of formulas in $prop(\psi)$ that are defined only over signals in H. Formally, $cl_H(\psi) = \bigcup_{\theta \in prop(\psi)} cl_H(\theta)$, where $cl_H(\theta)$ is defined for a propositional formula θ as follows.

- If θ is only over signals in H, then $cl_H(\theta) = \{\theta\}$.
- If θ is only over signals in $V \cup O$, then $cl_H(\theta) = \emptyset$.
- Otherwise, θ is of the form $\neg\theta_1$ or $\theta_1 * \theta_2$ for $* \in \{\vee, \wedge\}$, in which case $cl_H(\theta) = cl_H(\theta_1)$ or $cl_H(\theta) = cl_H(\theta_1) \cup cl_H(\theta_2)$, respectively.

For example, if $H = I = \{i_1, i_2, i_3\}$ and $O = \{o\}$, then $cl_H((i_1 \vee i_2) \wedge (i_3 \vee o)) = \{i_1 \vee i_2, i_3\}$. Note that formulas in $cl_H(\psi)$ are over H, and that the relation $\approx_{cl_H(\psi)}$ is an equivalence relation on 2^H. We say that a program $p : M \times 2^H \to M \times 2^G$ is *tight* for ψ if for every memory $m \in M$ and two assignments $h, h' \in 2^H$, if $h \approx_{cl_H(\psi)} h'$, then $p(m, h) = p(m, h')$.

In Theorem 5.2 below, we argue that in the context of LTL synthesis, one can always restrict attention to tight programs (see Example 5.3 for an example for such a restriction).

Theorem 5.2 *If φ is (V, H, C, G)-realizable by a TGE, then it is (V, H, C, G)-realizable by a TGE that uses only programs tight for φ.*

Proof: Let $\mathcal{T} = \langle V, H, C, G, S, s_0, \delta, M, m_0, \tau\rangle$ be a TGE that realizes φ with arbitrary programs. We define a labeling function $\tau' : S \times 2^V \to 2^C \times \mathcal{P}_{M,H,G}$ that uses only programs tight for φ, and argue that the TGE $\mathcal{T}' = \langle V, H, C, G, S, s_0, \delta, M, m_0, \tau'\rangle$, obtained from $\mathcal{T}$ by replacing τ by τ', realizes φ.

Recall that the relation $\approx_{cl_H(\varphi)}$ is an equivalence relation on 2^H. Let $rep : 2^H \to 2^H$ map each assignment $h \in 2^H$ to an assignment that represents the $\approx_{cl_H(\varphi)}$-equivalence class of h; for example, we can define the representative assignment to be the minimal equivalent assignment according to some order on 2^H.

We define τ' such that for every $s \in S$ and $v \in 2^V$ with $\tau(s, v) = \langle c, p\rangle$, we have $\tau'(s, v) = \langle c, p'\rangle$, where the program p' is such that for all $m \in M$ and $h \in 2^H$, we have that $p'(m, h) = p(m, rep(h))$. Clearly, $\mathcal{T}'$ uses only tight programs. Indeed, all the assignments in the same equivalence class of $\approx_{cl_H(\varphi)}$ are mapped to the same program.

We continue and prove that $\mathcal{T}'$ realizes φ. Consider an input sequence $w_I = \langle v_1, h_1\rangle \cdot \langle v_2, h_2\rangle \cdot \langle v_3, h_3\rangle \cdots \in (2^V \times 2^H)^\omega$. Let $r = s_0 \cdot s_1 \cdot s_2 \cdots \in S^\omega$ be the run of $\mathcal{T}$ on w_I, and let $\langle c_1, p_1\rangle \cdot \langle c_2, p_2\rangle \cdot \langle c_3, p_3\rangle \cdots \in (2^C \times \mathcal{P}_{M,H,G})^\omega$ be the sequence of labels along the transitions of r. Thus, $\langle c_j, p_j\rangle = \tau(s_{j-1}, v_j)$ for all $j \geq 1$. Since $\mathcal{T}'$ differs from $\mathcal{T}$ only in the programs that τ' generates, the run r is also the run of $\mathcal{T}'$ on w_I, and the sequence of labels along the transitions in it is $\langle c_1, p'_1\rangle \cdot \langle c_2, p'_2\rangle \cdot \langle c_3, p'_3\rangle \cdots$, where $\langle c_j, p'_j\rangle = \tau'(s_{j-1}, v_j)$ for all $j \geq 1$. Hence, if we let $\langle m_j, d_j\rangle = p'_j(m_{j-1}, h_j)$ for all $j \geq 1$, then $w_G = d_1 \cdot d_2 \cdot d_3 \cdots \in (2^G)^\omega$ is sequence of assignments to the output signals in G that $\mathcal{T}'$ instructs the environment to perform when it reads w_I.

Consider now the input sequence $w'_I = \langle v_1, rep(h_1)\rangle, \langle v_2, rep(h_2)\rangle, \langle v_3, rep(h_3)\rangle, \ldots \in (2^V \times 2^H)^\omega$. Since w_I and w'_I agree on the input signals in V, and since δ and τ depend only on the input signals in V, the run of $\mathcal{T}$ on w'_I is also r, and the sequence of labels along r is also $\langle c_1, p_1\rangle, \langle c_2, p_2\rangle, \langle c_3, p_3\rangle, \ldots \in (2^C \times \mathcal{P}_{H,G})^\omega$. Thus if we define $m'_0 = m_0$ and $\langle m'_j, d'_j\rangle = p_j(m'_{j-1}, rep(h_j))$, for all $j \geq 1$, then $w'_G = d'_1 \cdot d'_2 \cdot d'_3 \cdots \in (2^G)^\omega$ is the sequence of assignments to the output signals in G that $\mathcal{T}$ instructs the environment to perform when it reads w'_I.

We prove that $\langle m_j, d_j\rangle = \langle m'_j, d'_j\rangle$ for all $j \geq 1$. Recall that by definition of τ', for all $m \in M$ and $h \in 2^H$, we have that $p'_j(m, h) = p_j(m, rep(h))$. Hence, for all $j \geq 1$ we have $\langle m'_j, d'_j\rangle = p_j(m'_{j-1}, rep(h_j)) = p'_j(m'_{j-1}, h_j)$. Thus, as $m'_0 = m_0$, it follows by induction that $\langle m_j, d_j\rangle = p'_j(m'_{j-1}, h_j) = p'_j(m_{j-1}, h_j) = \langle m'_j, d'_j\rangle$, for all $j \geq 1$. In particular, $w'_G = w_G$, and so $\mathcal{T}$ generates on w'_I the same assignments to the signals in G as $\mathcal{T}'$ generates on w_I.

Thus, for the input sequence w'_I, the TGE $\mathcal{T}$ generates the computation $\mathcal{T}(w'_I) = \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdots \in (2^{I \cup O})^\omega$, with $\sigma_j = v_j \cup rep(h_j) \cup c_j \cup d_j$, for all $j \geq 1$. Since $\mathcal{T}$ realizes φ, we know that $\pi \models \varphi$. Then, for the input sequence w_I, the TGE $\mathcal{T}'$ generates the computation $\mathcal{T}'(w_I) = \sigma'_1 \cdot \sigma'_2 \cdot \sigma'_3 \cdots \in (2^{I \cup O})^\omega$, with $\sigma'_j = v_j \cup h_j \cup c_j \cup d_j$, for all $j \geq 1$.

Note that the computations $\mathcal{T}(w'_I)$ and $\mathcal{T}'(w_I)$ agree on $V \cup O$, and also agree on all the formulas in $cl_H(\varphi)$. Hence, as all the formulas in $prop(\varphi)$ are composed of predicates over $V \cup O \cup cl_H(\varphi)$, we have that $\mathcal{T}(w'_I) \approx_{prop(\varphi)} \mathcal{T}'(w_I)$. By Proposition 5.1, we thus have that $\mathcal{T}'(w_I) \models \varphi$, and we are done. □

Example 5.3 [Simplifying programs] Let $H = \{i_1, i_2\}$ and $G = \{o\}$. Consider the propositional specification $\varphi = (i_1 \wedge i_2) \leftrightarrow \neg o$. Consider the program $p : 2^H \to \{T, F\}$ that assigns values to o as follows (note that in a setting with no memory and a single guided output signal, programs are indeed of this type).

i_1	i_2	$p(i_1, i_2)$
F	F	T
F	T	T
T	F	F
T	T	T

Let $\theta = i_1 \wedge i_2$. Note that $cl_H(\varphi) = \{\theta\}$, and so p above is not tight, as $\{i_1\} \approx_{\{\theta\}} \{i_2\}$, yet $p_1(\{i_1\}) \neq p_1(\{i_2\})$. Also, while there are 16 programs in $\mathcal{P}_{\{i_1,i_2\},\{o\}}$, there are only four tight programs in it:

θ	$p_1(\theta)$	$p_2(\theta)$	$p_3(\theta)$	$p_4(\theta)$
F	F	F	T	T
T	F	T	F	T

Among these tight programs, program p_3 satisfies the specification, in the sense that φ is valid for all assignments in 2^I. Indeed, program p_3 induces the following

assignments:

i_1	i_2	θ	o	φ
F	F	F	T	T
F	T	F	T	T
T	F	F	T	T
T	T	T	F	T

Thus, φ can be realized by a TGE that uses a tight program.

Theorem 5.2 enables us to replace the domain $M \times 2^H$ by the more restrictive domain $M \times 2^{cl_H(\varphi)}$. Below we discuss how to reduce the domain further, taking into account the configurations in which the programs are going to be executed.

Recall that whenever a TGE instructs the environment which program to follow, the signals in V and C already have an assignment. Indeed, $\tau : S \times 2^V \to 2^C \times \mathcal{P}_{M,H,G}$, and when $\langle c, p\rangle \in \delta(s, v)$, we know that the program p is going to be executed when the signals in V and C are assigned v and c. I

Below we discuss and demonstrate how the known assignments to V and C can be used to make the equivalence classes of 2^H coarser. Note that the formulas in $cl_H(\theta)$ are over H, thus the assignment to the signals in $V \cup C$ does not affect them. Rather, it can "eliminate" from $prop(\varphi)$ formulas that are evaluated to `T` or `F`. For example, if $prop(\varphi) = \{(v_1 \wedge h_1 \wedge h_3) \vee d_1, (c_1 \vee h_2) \wedge d_2\}$, then an assignment of `T` to v_1 and c_1 simplifies the formulas to $\{(h_1 \wedge h_3) \vee d_1, d_2\}$. Formally, we have the following.

For a set Θ of propositional formulas over $I \cup O$ and an assignment $f \in 2^{V \cup C}$, let $\Theta_{|f}$ be the set of non-trivial (that is, different from `T` or `F`) propositional formulas over $H \cup G$ obtained by simplifying the formulas in Θ according to the assignment f to the signals in $V \cup C$. For example, if $\Theta = \{(v_1 \wedge h_1) \vee d_1, (c_1 \vee h_2) \wedge d_2\}$, and $f(v_1) = f(c_1) = \mathtt{F}$, then $\Theta_{|f} = \{d_1, h_2 \wedge d_2\}$.

For an LTL formula ψ, and an assignment $f \in 2^{V \cup C}$, let $cl_{H,f}(\psi)$ be the set of maximal subformulas of formulas in $prop(\psi)_{|f}$ defined only over signals in H. Formally, $cl_{H,f}(\psi) = \bigcup_{\theta \in prop(\psi)_{|f}} cl_H(\theta)$. For example, for φ with $prop(\varphi) = \{(v_1 \wedge h_1 \wedge h_3) \vee d_1, (c_1 \vee h_2) \wedge d_2\}$, while $cl_H(\varphi) = \{h_1 \wedge h_3, h_2\}$, we have the simplification presented in the table below.

$f(v_1)$	$f(c_1)$	$cl_{H,f}(\varphi)$
F	F	$\{h_2\}$
F	T	$\emptyset$
T	F	$\{h_1 \wedge h_3, h_2\}$
T	T	$\{h_1 \wedge h_3\}$

Now, we say that a program $p : M \times 2^H \to M \times 2^G$ is *f-tight* if for every memory $m \in M$ and two assignments $h, h' \in 2^H$, if $h \approx_{cl_{H,f}(\varphi)} h'$, then $p(m, h) = p(m, h')$.

Then, a TGE is tight if for every state $s \in S$ and assignments $v \in V$ with $\delta(s, v) = \langle c, p\rangle$, we have that p is $(v \cup c)$-tight. It is easy to extend the proof of Theorem 5.2 to tight TGEs. Indeed, now, for every position $j \geq 1$ in

the computation, the equivalence class of $h_j \in 2^H$ is defined with respect to $cl_{H,v_j \cup c_j}(\varphi)$, and all the considerations stay valid.

Remark 5.1 Typically, a specification to a synthesized system is a conjunction of sub-specifications, each referring to a different functionality of the system. Consequently, the assignment to each output signal may depend only on a small subset of the input signals – these that participate in the sub-specifications of the output signal. For example, in the specification $\varphi = \mathbf{G}(o_1 \leftrightarrow i_1) \wedge \mathbf{G}(o_2 \leftrightarrow i_2)$, the two conjuncts are independent of each other, and so the assignment to o_1 can depend only on i_1, and similarly for i_1 and o_2. Accordingly, further reductions to the set of programs can be achieved by decomposing programs to sub-programs in which different subsets of H are considered when assigning values to different subsets of G. In addition, by analyzing dependencies within each sub-specification, the partition of the output signals and their corresponding "affecting sets of hidden signals" can be refined further, and as showed in [1], finding dependent output signals may lead to improvements in state of the art synthesis algorithms. □

6 Viewing a TGE as a Distributed System

Consider a TGE $\mathcal{T}$ with memory M that (V, H, C, G)-realizes a specification. As shown in Figure 3 (left), the TGE's operation can be viewed as two distributed processes executed together: the TGE $\mathcal{T}$ itself, and a transducer $\mathcal{T}_G$ with state space M, implementing $\mathcal{T}$'s instructions to the environment. In each round, the transducer $\mathcal{T}_G$ receives from the environment an assignment to the signals in H, received from $\mathcal{T}$ a program, and uses both in order to generate an assignment to the signals in G and update its state.

Figure 3: A TGE that interacts with a guided environment (left) and the corresponding distributed system architecture (right).

We examine whether viewing TGEs this way can help reduce SGE to known algorithms for the synthesis of distributed systems. We argue that the approach here, where we do not view $\mathcal{T}_G$ as a process in a distributed system, is preferable. In distributed-systems synthesis, we are given a specification φ and an *architecture* describing the communication channels among processes. The goal is to design strategies for these processes so that their joint behavior satisfies φ. Synthesis of distributed systems is generally undecidable [28], primarily due to *information forks* – processes with incomparable information (e.g., when the environment sends assignments of disjoint sets of signals to two processes) [32].

The SGE setting corresponds to the architecture in Figure 3: Process P_1 is the TGE $\mathcal{T}$, which gets assignments to V and generates assignments to C. Instead of designing P_1 to generate instructions for the environment, the synthesis algorithm also returns P_2, which instructs the environment on generating assignments to G. The process P_2 gets (in fact generates) assignments to both V and H, eliminating information forks, making SGE solvable by solving distributed-system synthesis for this architecture. A solution in the TGE setting, that is composed of a TGE $\mathcal{T}$ and an environment transducer $\mathcal{T}_G$, induces a solution in the distributed setting: P_1 follows $\mathcal{T}$, and P_2 simulates the joint operation of $\mathcal{T}$ and $\mathcal{T}_G$, assigning values to G as instructed by $\mathcal{T}$. Conversely, a TGE can encode through $\mathcal{P}_{M,H,G}$ the current assignment to V together with a description of the structure of P_2, achieving the architecture in Figure 3 (right).

Using programs in $\mathcal{P}_{M,H,G}$ goes beyond sending V's values, which are already known to the environment. Programs leverage the TGE's computation, particularly its current state, to save resources and utilize less memory. Not using the communication channel between the TGE and the environment could result in a significant increase in the size of process P_2. For example, when $|M| = 1$ (specifications where G's assignment depends only on V's history and H's current assignment), the process P_2 is redundant. An explicit example is in Theorem 6.1, similar to the proof of Theorem 3.3. As demonstrated in Section 5, programs in $\mathcal{P}_{M,H,G}$ can be described symbolically. Formally, we have the following.

Theorem 6.1 *For every $k \geq 1$, there exists a specification φ_k over $V \cup H \cup G$, such that φ_k is (V, H, C, G)-realizable by a TGE with a set of states and a memory set both of size $O(k)$, yet the size of P_2 in a distributed system that realizes φ_k is at least $\Omega(k^2)$.*

Proof: Let $V = \{v\}, H = \{h\}$, and $G = \{d\}$. Consider the specification $\varphi_k = Fd \leftrightarrow ((F^k v) \wedge (F^k h))$, where the operator F^k stands for "at least k occurrences in the future" (see formal definition in Theorem 3.3). Accordingly, the specification φ_k requires that d is eventually turned on iff both signals v and h are turned on at least k times.

We prove that in the distributed setting, Process P_2 needs to implement two k-counters simultaneously, while a TGE may decompose the two counters between the system and the environment, implying the stated quadratic saving.

Note that the synthesis of φ_k in a distributed setting forces the process P_2 to implement two k-counters, one for the number of times v is on, and one for the number of times h is on. Accordingly, P_2 needs at $\Omega(k^2)$ states. On the other hand, synthesis of φ_k by a TGE enables a decomposition of the two counters: The TGE $\mathcal{T}$ maintains a counter for v, and the environment transducer $\mathcal{T}_G$ needs to implement only a counter for h. Indeed, as long that the v counter has a value below k, the TGE $\mathcal{T}$ sends to $\mathcal{T}_G$ a program that instruct it to assign d with `F`, and update its state according to h. Once the counter of v reaches k, the TGE instructs $\mathcal{T}_G$ to turn d on if its counter of h reached k. Thus, there is a TGE with state space and memory set both linear in k. □

7 Discussion

We introduced *synthesis with guided environments*, where the system can utilize the knowledge and computational power of the environment. Straightforward directions for future research include extensions of the many settings in which synthesis has been studied to the "the guided paradigm". Here we discuss two directions that are more related to the paradigm itself.

Dynamic hiding and guidance. In the setting studied here, the partition of I and O into visible, hidden, controlled, and guided signals is fixed throughout the computation. In some settings, these partitions may be dynamic. For example, when visibility depends on sensors that the system may activate and deactivate [2] or when signals are sometimes hidden in order to maintain the privacy of the system and the environment [21]. The decision which signals to hide in each round may depend on the system (e.g., when it instructs the environemnt which signals to hide in order to maintain its privacy), the environment (e.g., when it prefers not to share sensitive information), or an external authority (e.g., when signals become hidden due to actual invisibility). As for output signals, their guidance may depend on the history of the interaction (e.g., we may be able to assume amenability from the environment only after some password has been entered).

Bounded SGE. SGE involves a memory that can be used by the environment. As in the study of traditional *bounded synthesis* [33, 22], it is interesting to study SGE with given bounds on both the state spaces of the system and the environment. In addition to better modeling the setting, the bounds are used in order to improve the complexity of the algorithm, and they can also serve in heuristics, as in SAT-based algorithms for bounded synthesis [13]. In the setting of SGE, it is interesting to investigate the tradeoffs among the three involved bounds. It is easy to see that the two bounds that are related to the environment, namely the bound on its state space and the bound on the memory supervised by the system, are dual: an increase in the memory supervised by the system makes more specifications realizable, whereas an increase in the size of the state space of the environment makes fewer specifications realizable.

Another parameter that is interesting to bound is the number of different programs that a TGE may use, or the class of possible programs. In particular, restricting SGE to programs in which guided output signals can be assigned only the values of hidden signals or values stored in registers, will simplify an implementation of the algorithm. Likewise, the update of the memory during the interaction may be global and fixed throughout the computation.

References

[1] Akshay, S., Basa, E., Chakraborty, S., Fried, D.: On dependent variables in reactive synthesis. In: Proc. 30th Int. Conf. on Tools and Algorithms for

the Construction and Analysis of Systems. pp. 123–143. Springer Nature Switzerland (2024)

[2] Almagor, S., Kuperberg, D., Kupferman, O.: Sensing as a complexity measure. Int. J. Found. Comput. Sci. **30**(6-7), 831–873 (2019)

[3] Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5), 672–713 (2002)

[4] Anand, A., Mallik, K., Nayak, S., Schmuck, A.K.: Computing adequately permissive assumptions for synthesis. In: Proc. 29th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. pp. 211–228. Springer (2023)

[5] Berthon, R., Maubert, B., Murano, A., Rubin, S., Vardi, M.: Strategy logic with imperfect information. In: Proc. 32nd ACM/IEEE Symp. on Logic in Computer Science. pp. 1–12 (2017)

[6] Bloem, R., Chatterjee, K., Jobstmann, B.: Graph games and reactive synthesis. In: Handbook of Model Checking., pp. 921–962. Springer (2018)

[7] Bloem, R., Ehlers, R., Könighofer, R.: Cooperative reactive synthesis. In: 13th Int. Symp. on Automated Technology for Verification and Analysis. pp. 394–410 (2015)

[8] Bouyer, P., Markey, N., Vester, S.: Nash equilibria in symmetric graph games with partial observation. Information and Computation **254**, 238–258 (2017)

[9] Chatterjee, K., Doyen, L., Henzinger, T., Raskin, J.F.: Algorithms for omega-regular games with imperfect information. Logical Methods in Computer Science **3**(3) (2007)

[10] Chatterjee, K., Doyen, L., Nain, S., Vardi, M.: The complexity of partial-observation stochastic parity games with finite-memory strategies. In: Proc. 17th Int. Conf. on Foundations of Software Science and Computation Structures. Lecture Notes in Computer Science, vol. 8412, pp. 242–257. Springer (2014)

[11] Chatterjee, K., Majumdar, R., Henzinger, T.A.: Controller synthesis with budget constraints. In: Proc 11th International Workshop on Hybrid Systems: Computation and Control. Lecture Notes in Computer Science, vol. 4981, pp. 72–86. Springer (2008)

[12] De Giacomo, G., Vardi, M.Y.: Ltl_f and ldl_f synthesis under partial observability. In: Proc. 25th Int'l Joint Conf. on Artificial Intelligence. pp. 1044–1050. IJCAI/AAAI Press (2016)

[13] Ehlers, R.: Symbolic bounded synthesis. In: Proc. 22nd Int. Conf. on Computer Aided Verification. Lecture Notes in Computer Science, vol. 6174, pp. 365–379. Springer (2010)

[14] Esparza, J., Kretínský, J., Sickert, S.: A unified translation of linear temporal logic to ω-automata. J. ACM **67**(6), 33:1–33:61 (2020)

[15] Filiot, E., Gentilini, R., Raskin, J.F.: Rational synthesis under imperfect information. In: Proc. 33rd ACM/IEEE Symp. on Logic in Computer Science. pp. 422–431. ACM (2018)

[16] Finkbeiner, B., Metzger, N., Moses, Y.: Information flow guided synthesis with unbounded communication. In: Proc. 36th Int. Conf. on Computer Aided Verification. pp. 64–86. Springer Nature Switzerland (2024)

[17] Gurevich, Y., Harrington, L.: Trees, automata, and games. In: Proc. 14th ACM Symp. on Theory of Computing. pp. 60–65. ACM Press (1982)

[18] Gutierrez, J., Perelli, G., Wooldridge, M.J.: Imperfect information in reactive modules games. Inf. Comput. **261**, 650–675 (2018)

[19] Krausz, A., Rieder, U.: Markov games with incomplete information. Mathematical Methods of Operations Research **46**, 263–279 (1997)

[20] Kumar, R., Shayman, M.: Formulae relating controllability, observability, and co-observability. Autom. **34**(2), 211–215 (1998)

[21] Kupferman, O., Leshkowitz, O.: Synthesis of privacy-preserving systems. In: Proc. 42nd Conf. on Foundations of Software Technology and Theoretical Computer Science. Leibniz International Proceedings in Informatics (LIPIcs), vol. 250, pp. 42:1–42:23 (2022)

[22] Kupferman, O., Lustig, Y., Vardi, M., Yannakakis, M.: Temporal synthesis for bounded systems and environments. In: Proc. 28th Symp. on Theoretical Aspects of Computer Science. pp. 615–626 (2011)

[23] Kupferman, O., Vardi, M.: Synthesis with incomplete information. In: Advances in Temporal Logic. pp. 109–127. Kluwer Academic Publishers (2000)

[24] Kupferman, O., Vardi, M.: Synthesizing distributed systems. In: Proc. 16th ACM/IEEE Symp. on Logic in Computer Science. pp. 389–398 (2001)

[25] Kupferman, O., Vardi, M.: Safraless decision procedures. In: Proc. 46th IEEE Symp. on Foundations of Computer Science. pp. 531–540 (2005)

[26] Pnueli, A.: The temporal semantics of concurrent programs. Theoretical Computer Science **13**, 45–60 (1981)

[27] Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. 16th ACM Symp. on Principles of Programming Languages. pp. 179–190 (1989)

[28] Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proc. 31st IEEE Symp. on Foundations of Computer Science. pp. 746–757 (1990)

[29] Reif, J.: The complexity of two-player games of incomplete information. Journal of Computer and Systems Science **29**, 274–301 (1984)

[30] Rosner, R.: Modular Synthesis of Reactive Systems. Ph.D. thesis, Weizmann Institute of Science (1992)

[31] Safra, S.: On the complexity of ω-automata. In: Proc. 29th IEEE Symp. on Foundations of Computer Science. pp. 319–327 (1988)

[32] Schewe, S.: Synthesis of distributed systems. Ph.D. thesis, Saarland University, Saarbrücken, Germany (2008)

[33] Schewe, S., Finkbeiner, B.: Bounded synthesis. In: 5th Int. Symp. on Automated Technology for Verification and Analysis. Lecture Notes in Computer Science, vol. 4762, pp. 474–488. Springer (2007)

[34] Vardi, M., Wolper, P.: Automata-theoretic techniques for modal logics of programs. Journal of Computer and Systems Science **32**(2), 182–221 (1986)

[35] Vardi, M., Wolper, P.: Reasoning about infinite computations. Information and Computation **115**(1), 1–37 (1994)

[36] Wu, Y., Raman, V., Rawlings, B., Lafortune, S., Seshia, S.: Synthesis of obfuscation policies to ensure privacy and utility. Journal of Automated Reasoning **60**(1), 107–131 (2018)