On the Complexity of Realizability for Safety LTL and Related Subfragments

Noel Arteche
Department of Computer Science, University of the Basque Country, Spain

Montserrat Hermo
Department of Computer Science, University of the Basque Country, Spain

Abstract
We study the realizability problem for Safety LTL, the syntactic fragment of Linear Temporal Logic capturing safe formulas. We show that the problem is \( \text{EXP}\)-complete, disproving the existing conjecture of \( 2\text{EXP}\)-completeness. We achieve this by comparing the complexity of Safety LTL with seemingly weaker subfragments. In particular, we show that every formula of Safety LTL can be reduced to an equirealizable formula of the form \( \alpha \land \Box \psi \), where \( \alpha \) is a present formula over system variables and \( \psi \) contains Next as the only temporal operator. The realizability problem for this new fragment, which we call \( \text{GX}_0 \), is also \( \text{EXP}\)-complete.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Problems, reductions and completeness

Keywords and phrases temporal logic, LTL, realizability, Safety LTL, complexity, equirealizability

Funding Funding for this work has been provided by the European Union (ERDF funds) under the grant PID2020-112581GB-C22 and by the University of the Basque Country under the LoRea GIU18-182 Project.

1 Introduction

The realizability and synthesis problem for logical specifications is a well-researched topic in computational logic. Originally introduced by Church [5] and later applied to the context of Linear Temporal Logic (LTL) specifications, it considers a logical formula where some variables belong to the system and some to the environment, and asks the system to produce a winning strategy so that no matter what the environment does, the system is able to satisfy the formula.

The problem is of central importance in the correctness verification of reactive systems and has hence received a great deal of attention. Unfortunately, the \( 2\text{EXP}\)-completeness of realizability for LTL specifications [14] renders infeasible in practice to work with the full expressive power of Linear Temporal Logic. Instead, researchers have focused their efforts on developing algorithms for certain subfragments capable of expressing useful specifications while achieving provably lower complexity bounds.

Let us briefly highlight some of the contributions to this line of research. Starting in 2004, Alur and La Torre gave in [1] a survey of the complexities of realizability for some subfragments of LTL obtained by simply adding and removing temporal operators. In an attempt to work with a more structured fragment, Piterman et al. introduced in 2006 the Generalized Reactivity(1) formulas, GR(1) for short. As shown in their seminal paper [13], realizability for GR(1) can be solved in time exponential in the size of the specification. Besides, Ehlers later proved that realizability for this fragment is actually complete for \( \text{EXP} \) under polynomial-time reductions [10].

In an attempt to go even lower in the complexity map, Cheng et al. designed in 2016 the GXW fragment [4]. In GXW specifications we are allowed to use the Globally, Next and Weak-Until operators, hence capturing a wide variety of practical scenarios, but only in
formulas adhering to very specific patterns. This trade-off between the available operators and tight structure in formulas led to an elaborate algorithm that brought the complexity of realizability for GXW formulas down to PSPACE (without completeness known to date).

More recently, the interest focused on fragments capable of expressing safety properties. In this area, the most interesting one is Safety LTL, the Until-free fragment of LTL in Negation Normal Form [17]. Safety LTL has been proven to capture the semantic notion of safety expressible in LTL [7], turning it into a central fragment of temporal logic for realizability purposes. For these formulas, Zhu et al. gave in [17] an algorithm that while running in double exponential time, attained decent running times in practice. However, they paid no attention to structural complexity considerations.

In an attempt to improve the double exponential complexity of the existing algorithms for Safety LTL, Cimatti et al. introduced in [6] the Extended Bounded Response fragment, LTL_{EBR}, a syntactic subfragment of Safety LTL, as well as LTL_{EBR} + P, consisting of LTL_{EBR} extended with past operators. They provided an exponential-time algorithm for the realizability problem on these formulas and showcased efficient empirical results. Furthermore, they showed in [8] that realizability for LTL_{EBR} and LTL_{EBR} + P is EXP-complete.

While LTL_{EBR} is a syntactic fragment of Safety LTL, it is not expressive enough to capture all of Safety LTL. This was shown in [7], where they proved that LTL_{EBR} is strictly less expressive than LTL_{EBR} + P, which is in turn as expressive as Safety LTL. Despite the expressive equivalence between Safety LTL and LTL_{EBR} + P, it does not follow that realizability for Safety LTL is EXP-complete. This is because, in principle, formulas in LTL_{EBR} + P could be exponentially more succinct than the Safety LTL ones, due to the past modalities.

When comparing fragments, the existing research focuses solely on providing logically equivalent formulas between fragments, over the same sets of atoms. In practice, however, the requirement of keeping strict logical equivalence can be dropped in favour of equirealizable fragments. We believe that the comparison between fragments should be carried out mainly in terms of complexity considerations and not in terms of expressibility. After all, as long as a formula in one fragment can be efficiently translated into an equirealizable formula in another fragment, possibly by introducing auxiliary atoms, the solution will be satisfactory.

In this paper, we study the realizability for Safety LTL from a purely complexity-theoretic perspective. We provide reductions between fragments, which leave behind logical equivalence in favour of equirealizability. In this way, our main result it to show that a very restricted fragment of LTL_{EBR}, which we call GX₀, is equirealizable to Safety LTL. By showing EXP-completeness for our restricted fragment GX₀, our reduction proves the EXP-completeness of realizability for Safety LTL, a problem conjectured to be 2EXP-complete. Our (main) fragment, GX₀, consists of formulas of the form α ∧ □ ψ, where α is a present formula (i.e., without temporal operators) over system variables and ψ is a formula where Next is the only possible temporal operator. Note that it is not obvious a priori that such a reduction is possible, since Safety LTL contains operators like Release, while our fragment is a lot more restricted. Crucially, our formulas are always of a fixed simple form, which makes it easier to develop algorithms for (in contrast with existing fragments like GR(1), LTL_{EBR} or GXW).

The structure of the paper is as follows. Section 2 recalls definitions from temporal logic as well as the grammar of Safety LTL, and introduces the three new fragments we study: X, GX and GX₀, the latter being the most important one. In Section 3 we study the complexity of realizability for these fragments, showing PSPACE-completeness for formulas in X and EXP-completeness for formulas in GX and GX₀. The latter is achieved via a reduction to
the acceptance problem for polynomial-space alternating Turing machines, while for \( GX \) we provide a polynomial-time reduction from \( GX_0 \). This reduction exploits our central technical lemma (Lemma 13), stating that one can remove the initial condition in \( GX_0 \) formulas to obtain an equirealizable \( GX \) specification. The lemma is also crucial to our main theorem (Theorem 15): realizability for Safety LTL is \( \text{EXP} \)-complete, proven via a reduction to \( GX_0 \). This is carried out in Section 4, concluding that \( GX_0 \) and Safety LTL are equirealizable fragments. Furthermore, the section compares \( GX_0 \) to the fragment \( \text{LTL}_{\text{EBR}} \) of Cimatti et al. from [6, 7, 8] and their expressibility results. Section 5 concludes with an overview of the results and some comments on the role of complexity theory in the future of temporal logic in particular and computational logic in general. Appendix A contains an explicit translation between the \( GXW \) specifications of [4] into equirealizable specifications of \( GX_0 \), further showcasing the simplicity and power of this method. A direct reduction from \( GXW \) into \( X \) (both of which are in PSPACE) remains an open problem.

2 Preliminaries

We denote by LTL the set of well-formed formulas of Linear Temporal Logic, given by the grammar
\[
\varphi ::= \bot \mid p \mid \neg \varphi \mid \varphi \land \varphi \mid \square \varphi \mid \varphi U \varphi
\]
where \( p \in \mathcal{P} \), some set of Boolean variables. A fragment of LTL is a subset of LTL.

The satisfaction relation for LTL formulas is defined over infinite binary words. Let \( \mathbb{B} = \{0, 1\} \). Given a formula, a trace or model \( \sigma \in (\mathbb{B}^\omega) \) and a point \( i \in \mathbb{N} \), we have
\[
\begin{align*}
\sigma_i \not\models \bot & \quad & \sigma_i \models \square \varphi & \text{iff } \sigma_{i+1} \models \varphi \\
\sigma_i \models p & \text{ iff } \sigma_i(p) = 1 & \sigma_i \models \varphi U \psi & \text{iff there is a } j \geq i \text{ such that } \sigma_j \models \psi \text{ and for all } i \leq k < j, \sigma_k \models \varphi \\
\sigma_i \models \varphi \land \psi & \text{ iff } \sigma_i \models \varphi \text{ and } \sigma_i \models \psi \\
\end{align*}
\]

We write \( \sigma \models \varphi \) for \( \sigma_0 \models \varphi \).

The syntax and semantics of LTL can be extended to include past operators. For our purposes, it suffices to consider the Yesterday (\( \odot \)) and Since (\( \mathcal{S} \)) operators:
\[
\begin{align*}
\sigma_i \models \odot \varphi & \text{ iff } i > 0 \text{ and } \sigma_{i-1} \models \varphi \\
\sigma_i \models \varphi \mathcal{S} \psi & \text{ iff there is a } j \leq i \text{ such that } \sigma_j \models \psi \text{ and for all } j < k \leq i, \sigma_k \models \varphi.
\end{align*}
\]

In this paper we do not handle formulas containing the Until (\( U \)) operator, though we do work with the related operators Globally (\( \square \varphi := \varphi U \bot \)), Eventually (\( \Diamond \varphi := \top U \varphi \)), Weak-Until (\( \varphi_1 W \varphi_2 := \square \varphi_1 \lor (\varphi_1 U \varphi_2) \)), and Release (\( \varphi_2 R \varphi_1 := \neg (\neg \varphi_2 U \neg \varphi_1) \)).

For conciseness, we write \( \odot^i \varphi \) for \( \varphi \) if \( i = 0 \) and \( \odot^{i-1} \varphi \) if \( i > 0 \). A formula \( \odot^i t \) for some literal \( t \) is a temporal atom. For \( a, b \in \mathbb{N}, a \leq b \), \( \square_{[a,b]} \varphi \) stands for \( \wedge_{i=a}^b \odot^i \varphi \). These are all metalinguistic abbreviations, and the problems considered always get as input an unfolded formula without bounded operators.

Finally, given a temporal formula \( \varphi \) with Next (\( \odot \)) as the only temporal operator, we define its temporal depth as the largest number of nested occurrences of Next.

Following [11], given a finite set of atoms \( \mathcal{P} \), a language \( \mathcal{L} \subseteq (\mathbb{B}^\omega)^\omega \) is called safe if every \( \sigma \in \mathcal{L} \) has a bad prefix: a finite word \( p \in (\mathbb{B}^\omega)^* \) such that for all \( \tau \in (\mathbb{B}^\omega)^\omega \), \( p \cdot \tau \not\in \mathcal{L} \). For a formula \( \varphi \), we denote by \( \mathcal{L}(\varphi) \) the set of models of \( \varphi \). A formula \( \varphi \) is safe if \( \mathcal{L}(\varphi) \) is safe.

In [17], Zhu et al. studied Safety LTL, a syntactic fragment of LTL intended to capture the safe formulas.
On the Complexity of Realizability for Safety LTL

Definition 1 (Safety LTL). We denote by Safety LTL the syntactic fragment of LTL determined by the grammar

\[ \varphi ::= p \mid \neg p \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \varnothing \varphi \mid \varnothing \varphi \]  

which corresponds to the Until-free fragment of LTL in Negation Normal Form (NNF).

For some fragment \( \text{Frag} \subseteq \text{LTL} \) of well-formed formulas, we define \( [\text{Frag}] := \{ \mathcal{L}(\varphi) \mid \varphi \in \text{Frag} \} \). Indeed, \( [\text{Safety LTL}] = \{ [\varphi \in \text{LTL} \mid \varphi \text{ is safe}] \} \) (see, for example, [15]).

We are interested in the problem of realizability for LTL specifications.

Definition 2 (Realizability). Let \( \varphi \) be an LTL formula over variables from \( \mathcal{E} \cup \mathcal{S} \). The variables in \( \mathcal{E} \) are called environment or input variables, while the ones in \( \mathcal{S} \) are system or output variables. We say that \( \varphi \) is realizable if there is a function \( s : (\mathbb{B}^\mathcal{E})^* \rightarrow \mathbb{B}^\mathcal{S} \), called a winning strategy, such that for every infinite sequence of environment assignments \( E_0, E_1, E_2, \ldots \in (\mathbb{B}^\mathcal{E})^* \), we have \( E_0 \cup s(E_0), E_1 \cup s(E_0, E_1), E_2 \cup s(E_0, E_1, E_2), \ldots \models \varphi \).

Whenever a formula \( \varphi \) is realizable, the task of computing the winning strategy is called synthesis. For the particular case when \( \varphi \) contains no environment variables, the problem reduces to satisfiability (which is PSPACE-complete, [14]), but is otherwise more general.

Example 3. Consider the formula \( \varphi = q \land \Box(p \leftrightarrow \Diamond q) \), assuming \( p \) to belong to the environment and \( q \) to the system. The formula is realizable: the winning strategy sets \( q \) to true at the initial state, and then checks the value of \( p \) the moment before to set \( q \) accordingly.

Example 4. Take the following formula, slightly different from \( \varphi \) in the example above: \( \psi = q \land \Box(\Diamond p \leftrightarrow q) \), where \( p \) is still an input atom and \( q \) an output atom. This formula is not realizable. Suppose it was. Then, the strategy needs to make \( q \) true at the initial state; the environment then plays \( p \rightarrow 0 \) at the following step to force the system to a contradiction.

Throughout the paper, we translate formulas from one fragment to formulas in another fragment. Since these fragments often have different expressive power, we cannot produce equivalent formulas. For our purposes, a simple notion of equirealizability suffices.

Definition 5 (Equirealizability). We say that two formulas \( \varphi \) and \( \psi \) are equirealizable and write \( \varphi \equiv \psi \) if it holds that \( \varphi \) is realizable if and only if \( \psi \) is realizable. For two fragments \( \text{Frag}_1, \text{Frag}_2 \subseteq \text{LTL} \), we write \( \text{Frag}_1 \equiv \text{Frag}_2 \) to indicate that they are equirealizable fragments: for every \( \varphi \in \text{Frag}_1 \), there is an equirealizable \( \psi \in \text{Frag}_2 \), and vice versa.

This is in contrast with the usual notion of logical equivalence used in the literature.

Definition 6 (Expressive equivalence). We say that two formulas \( \varphi \) and \( \psi \) are logically equivalent if \( \mathcal{L}(\varphi) = \mathcal{L}(\psi) \). Two fragments \( \text{Frag}_1, \text{Frag}_2 \subseteq \text{LTL} \) are expressively equivalent if for every \( \varphi \in \text{Frag}_1 \), there exists a logically equivalent formula \( \psi \in \text{Frag}_2 \), and vice versa, and we write \( [\text{Frag}_1] = [\text{Frag}_2] \).

Remark 7. Note that if two formulas \( \varphi \) and \( \psi \) are logically equivalent, then it is implicit that they are defined over the same set of atoms. This need not happen between equirealizable formulas, where one of them can be defined over a larger set of atoms. In general, equivalent formulas are also equirealizable, but the converse does not hold.

Starting from Safety LTL, we introduce three new subfragments of it.
Definition 8 (X, GX and GX₀). We denote by X the syntactic fragment of LTL consisting of formulas where the only temporal operator is Next (O).

We denote by GX the syntactic fragment of LTL consisting of formulas of the form □ψ, where ψ ∈ X.

Finally, we denote by GX₀ the syntactic fragment consisting of formulas of the form α ∧ □ψ, where ψ ∈ X and α is a present formula over system variables.

We denote by X-Real, GX-Real and GX₀-Real the problems of deciding whether a given specification in respectively X, GX and GX₀ is realizable.

3 Complexity of realizability for X, GX and GX₀

We start by discussing the complexity of the realizability problem for specifications in X, GX and GX₀. We show that these three fragments are robust enough to exhibit completeness for either PSPACE or EXP.

3.1 Complexity for X

The problem X-Real is the set of all realizable specifications ψ ∈ X. Formulas in X can only talk about a finite initial segment of time, but the embedded alternation between system and environment is strong enough to capture all the problems in PSPACE.

Theorem 9. X-Real is PSPACE-complete.

Proof. Note the close resemblance between a temporal formula ψ with Next-only and a quantified Boolean formula. That X-Real ∈ PSPACE follows from the observation that one can push negation and Next to the literals of ψ and then consider each temporal atom to be a literal in the matrix of a quantified Boolean formula, where we make explicit on the quantifier prefix that system variables are existential and environment variables are universal. That is, if ψ ∈ X uses variables from E = {x₁, ..., xₙ} and S = {y₁, ..., yₘ} and has temporal depth d, then one can obtain a QBF

\[ ∀x₁ \ldots xₙ ∃y₁ \ldots yₘ \ldots ∀O^{d}x₁ \ldots O^{d}xₙ ∃O^{d}y₁ \ldots O^{d}yₘ : ψ \]

whose satisfiability can be checked in polynomial space.

To show hardness, it suffices to see that a quantified Boolean formula Φ = ⃗Q : ϕ(x₁, ..., xₙ) can be immediately converted into a formula ψ ∈ X. For every variable xᵢ in ϕ, if xᵢ appears existentially quantified at level k, add it as a system variable and replace every occurrence of xᵢ with Oᵏxᵢ; analogously, if xᵢ appears universally quantified at level k, add it as an environment variable and replace every occurrence of xᵢ with Oᵏxᵢ. Then, Φ is satisfiable if and only if ψ is realizable.

Both reductions in the previous proof are rather immediate: the bounded temporal depth of a specification in X is just a syntactic rewriting of a QBF. It might seem like X-Real is quite restricted and one cannot model interesting scenarios in this fragment. Surprisingly, in [4], Cheng et al. introduced GXW, a fragment of LTL for which realizability is in PSPACE. In GXW, formulas may contain the Globally (□), Next (O) and Weak-Until (W) operators, but adhering to very strict formula patterns. The restricted structure of the problem leads to its lower complexity, while it still captures a wide range of practical industrial applications.

Arguably, a specification in X is conceptually simpler than a GXW one, and the fact that X-Real is PSPACE-complete implies that there is a reduction from the realizability
problem of GXW to X-REAL. Such a reduction is precisely a mapping between the fragments that preserves realizability. Hence, instead of following complicated algorithms like the one originally presented in the context of GXW, one could just translate the specification into X and then call a QBF solver.

The structure of GXW formulas is discussed in detail in Appendix A, where we present a way to translate them into GX₀ specifications. Unfortunately, we have been unable to provide a direct translation into X. The restricted nature of GXW also leads us to conjecture that its realizability problem is not complete for PSPACE.

3.2 Complexity for GX₀ and GX

The specifications in X we just covered are quite limited in that one can only talk about a finite initial segment of time. In practice, one often wants to specify some correctness property of a system, and then enforce that this correctness property always holds. Moreover, one would like the system to react to the environment according to the specification, so that decisions made by the environment could affect the system’s decisions in the next states.

This intuition naturally leads us to considering the fragment GX₀, consisting of specifications of the form \( \alpha \land \Box \psi \), with \( \psi \in X \) and \( \alpha \) a present formula over system variables. This fragment imposes an initial condition \( \alpha \) for the system and then enforces a given specification \( \psi \) with Next-only to be satisfied at all points in time, which coincides with the intuition presented above. While the general problem of deciding whether an LTL specification is realizable is 2EXP-complete, the problem for GX₀ is provably easier.

\begin{prop}
GX₀-REAL is in EXP.
\end{prop}

\begin{proof}
We use the fact that EXP = APSPACE [12], the class of problems computable by an alternating Turing machine in polynomial space, and instead show membership in APSPACE. Recall that an alternating Turing machine (ATM) is a non-deterministic Turing machine where each state is labeled either “existential” or “universal”. In order for a run to be accepting, when being in an existential state, there must exist a transition that eventually leads to an accepting state, while on universal states, all transitions must eventually lead to accepting states.

To show membership of GX₀-REAL in APSPACE, see that checking whether a given safety specification \( \alpha \land \Box \psi \in GX₀ \) with \( n := |S| + |E| \) variables is unrealizable can be computed by an alternating Turing machine in polynomial space. First, compute the temporal depth \( d \) of \( \psi \), which is just the largest number of nested occurrences of Next and is hence linear in the size of the formula. Using the universal steps, the environment will choose an assignment to the input variables. Then, using the existential steps, the system chooses an assignment so that \( \alpha \) and \( \psi \) are both satisfied, based on the environment’s choice. Then, on the universal steps, an assignment is made for the environment variables; using the existential steps, an assignment is made to the system variables again, and so on. The idea is to keep the machine running while the model being constructed satisfies the formula. As soon as the formula is unsatisfiable given the choices made, the machine accepts. In order for the model built between system and environment to respect the specification at future points in time, the machine keeps on its tape the last \( d \) variable assignments. This takes space \( O(n \cdot d) \). Hence, the machine uses polynomial space and accepts if and only if \( \alpha \land \Box \psi \) is unrealizable. We have that GX₀-REAL ∈ APSPACE, but since the class is closed under complementation, GX₀-REAL ∈ APSPACE.
\end{proof}

The previous membership result was already known, since GX₀ ⊆ LTL_{EBR} and the algorithm for LTL_{EBR} realizability presented in [8] runs in exponential time. Like most such
arguments in the literature, complexity is derived from a careful analysis of a rather con-
voluted algorithm, while one rarely finds proofs using abstract models of computation, like
the one above.

In the same work, Cimatti et al. further showed that realizability for $\text{LTL}_{EBR}$ is $\text{EXP}$-complete, but since $\text{GX}_0 \subsetneq \text{LTL}_{EBR}$, that result does not entail completeness for $\text{GX}_0$. To the best of our knowledge, the following hardness result is new.

**Proposition 11.** $\text{GX}_0$-$\text{REAL}$ is $\text{EXP}$-hard.

**Proof.** We use again the fact that $\text{EXP} = \text{APSPACE}$. To show hardness, consider the standard acceptance problem

$$\{ \langle A, 1^s \rangle | \text{ATM } A \text{ accepts the empty word in space } s \},$$

which is complete for $\text{APSPACE}$, and show a reduction from it to $\text{GX}_0$-$\text{REAL}$.

For the purpose of this proof, we consider a simplified definition of alternating Turing machines, where they contain just two transition functions, $\delta$ and $\delta'$. This simplified definition is the one used, for example, by Arora and Barak in [2]. In the existential states, at least one of the two transitions must eventually lead to an accepting state for the machine to accept, while on universal states both transitions must eventually lead to accepting states. Hence, let $A = \langle Q, \delta, \delta', q_0 \rangle$ be an ATM, where $\delta, \delta' : Q \times \mathbb{B} \rightarrow Q \times \mathbb{B} \times \{1, -1\}$ and we assume $Q$ to be partitioned into $Q_\exists \cup Q_\forall$, so that states are either universal or existential, with initial state $q_0 \in Q$, accepting state $q_{\text{Accept}}$ and rejecting state $q_{\text{Reject}}$. We assume the machine to have a single tape, working in binary, extending infinitely to the right, where positions are numbered starting at 1. Because we only allow $s$ cells of space, we can consider any move above position $s$ to be illegal.

Note that if $A$ does not accept the empty word in space $s$, that is because either it rejects the empty word in space $s$; it never halts, but never exceeds space $s$; or it uses more than $s$ cells of memory. We will write a specification $\alpha \land \Box \psi$ that becomes trivially unrealizable for the system as soon the machine accepts without exceeding the space bound, and stays realizable in any of the previous three cases.

Let us specify the variables of the formulas $\alpha$ and $\psi$. The system will own:

- for every $i \in \{1, \ldots, s + 1\}$ and every $q \in Q$, a variable $p_{i,q}$ expressing that the machine is at state $q$ reading position $i$ of the tape;
- for every $i \in \{1, \ldots, s\}$, a variable $c_i$ expressing whether the $i$-th cell contains a 0 or a 1;
- a variable $\delta_e$ to determine whether transition function $\delta$ or $\delta'$ was chosen by the system.

The environment will own a single variable $\delta_e$, intended to determine whether transition function $\delta$ or $\delta'$ was chosen at a universal state.

Due to space limitations, we refer the reader to Appendix B for the explicit formulas, while here we limit ourselves to describing their intended meaning.

First, consider a Boolean formula $\alpha$, expressing that at the beginning, the machine starts at position 1 and state $q_0$ and that the tape is all zeroes. Then, we can write a temporal formula $\psi$ where $\text{Next}$ is the only temporal operator expressing the following constraints:

1. The machine must be in one and only one state, reading one and only one cell at a time.
2. If the current state is illegal (i.e., either the rejecting state or above the space limits), then keep the state of the machine unchanged for the next state.
3. At existential states, the system must choose a transition function ($\delta$ or $\delta'$) and the state of the machine must be updated accordingly for the next state, and analogously for universal states.
Let $\psi := S \land I \land T \land F$. This completes the specification $\alpha \land \Box \psi$. If at some point the machine accepts, then the specification immediately becomes false, because $F$ is falsified. On the other hand, if the machine exceeds space bounds or rejects, then the specification is trivially realizable for the system, because it can stay in the same state forever.

We have that $A$ accepts the empty word in space $s$ if and only if $\alpha \land \psi$ is unrealizable. This is a reduction from the acceptance problem to $\text{GX}_0\text{-REAL}$. Since the class is closed under complementation, we also have hardness for $\text{GX}_0\text{-REAL}$.

The previous two propositions prove the completeness result, as desired.

\textbf{Corollary 12.} $\text{GX}_0\text{-REAL}$ is $\text{EXP}$-complete.

Interestingly, a small modification to $\text{GX}_0$ formulas yields an even stronger result: we can get rid of the initial condition $\alpha$ and keep the same complexity. This corresponds to the problem $\text{GX-REAL}$, deciding whether a formula $\Box \psi \in \text{GX}$ is realizable. The complexity turns out to be the same, using the following lemma.

\textbf{Lemma 13.} For every $\alpha \land \Box \psi \in \text{GX}_0$, there is an equirealizable $\Box \chi \in \text{GX}$ that can be efficiently obtained from $\alpha \land \Box \psi$.

\textbf{Proof.} Let $r$ be a new environment variable that does not occur in $\psi$. The idea is that $r$ can “reset” the game. That is, if the environment sets $r$ to true, then the system must start over by satisfying $\alpha$, while as long as $r$ is set to false, the game keeps running. An initial attempt to translate $\alpha \land \psi$ is to consider the formula

$$\Box (r \rightarrow (\alpha \land \psi) \land \neg r \rightarrow \psi)$$

which captures the intuition, but fails to work. For example, the $\text{GX}_0$ formula $p \land \Box (p \leftrightarrow \Box \neg p)$ is realizable if $p$ is a system variable, but the proposed translation yields

$$\Box (r \rightarrow (p \land (p \leftrightarrow \Box \neg p)) \land \neg r \rightarrow (p \leftrightarrow \Box \neg p))$$

which is unrealizable (it suffices for the environment to reset twice in a row).

To fairly give the system a chance to recover from the reset, we need to wait some time, so that the current truth-assignment has really no effect. Suppose $\psi$ has depth $d$. Then, we should wait at least $d$ steps in time before we ask the system to make $\alpha \land \psi$ true again. With this in mind, we claim that the formula

$$\Box \left( (\neg r \rightarrow \psi) \land (\square [0,d]r \land \Box^{d+1} \neg r \rightarrow \Box^{d+1} \alpha) \right)$$

is equirealizable to $\alpha \land \Box \psi$.

Intuitively, the formula states that if the environment wishes to reset the game, it must set $r \mapsto 1$ for $d$ steps in a row, followed by $r \mapsto 0$. While that happens, the system is exempted from satisfying anything.

We now argue that equirealizability holds. For the forward direction, suppose that $\alpha \land \Box \psi$ is realizable by strategy $s : (B^F)^* \rightarrow B^S$. Then, we want to define a strategy $s' : (B^{F \cup \{r\}})^* \rightarrow B^S$ that extends $s$ to the new setting. Intuitively, if the environment never rests the game, then $s'$ should simply follow $s$; if the environment resets (by setting $r \mapsto 1$ for $d$ steps in a row, followed by $r \mapsto 0$), then the system should play with $s$,
pretending that the last move of the system is the initial one. That is, for every finite sequence $E_0, \ldots, E_n \in (\mathbb{B}^{E \cup \{r\}})^*$, define $s'$ as

$$s'(E_0, \ldots, E_n) := s(E_k \mathbin{|}_{E}, \ldots, E_n \mathbin{|}_{E}),$$

where $k$ is the largest $i \leq n$ such that $E_{i-d-1}(r) = \cdots = E_{i-1}(r) = 1$ and $E_i(r) = 0$, or 0 if such a $k$ does not exist. That is, $k$ is the last time a full resetting sequence was completed.

Observe that the strategy $s'$ is winning for the system over the new formula. We distinguish three possible scenarios:

(a) If the environment never restarts the game (i.e., always plays $r \mapsto 0$), then the system has to realize $\Box \psi$. In this case, the strategy $s'$ is just $s$, so by realizing $\alpha \land \Box \psi$, the system is also realizing $\psi$.

(b) If the environment plays $r \mapsto 1$ at some point, but not for $d$ steps in a row, then the strategy $s'$ will keep playing as if nothing happened. At the moments where $r \mapsto 1$, the entire new formula will be vacuously true for the system, so it cannot lose, and the strategy will keep playing just as if it was expected to satisfy $\psi$.

(c) Finally, if at some point the environment plays $r \mapsto 1$ for $d$ or more steps, followed by $r \mapsto 0$, then $s'$ resets the strategy, as if this was the beginning of the game. Note that the sudden shift in the truth-assignment for the system cannot possibly affect the game. This is because for the last $d$ or more steps, the new formula became vacuously true for the system.

Hence, if $\alpha \land \Box \psi$ is realizable by strategy $s$, so is the new formula by strategy $s'$. For the backwards direction, if the new formula is realizable by strategy $t$, then we can extract a winning strategy $t'$ for the original formula by checking what $t$ does when resetting the game just once, at the beginning. More formally, let $R := \{v \mapsto 1 \mid v \in E \cup \{r\}\} \in \mathbb{B}^{E \cup \{r\}}$ be a default “resetting assignment”. Then, the strategy $t' : (\mathbb{B}^E)^* \rightarrow \mathbb{B}^S$ defined as

$$t'(E_0, \ldots, E_n) := t(R, \ldots, R, E_0 \cup \{r \mapsto 0\}, \ldots, E_n \cup \{r \mapsto 0\})$$

is winning for the system over $\alpha \land \Box \psi$. Therefore, both formulas are equirealizable.

\begin{theorem}
GX-Real is EXP-complete.
\end{theorem}

\begin{proof}
Membership in EXP follows from the fact that GX is a syntactic fragment of GX_0, so the procedure in Proposition 10 also decides realizability for GX. For hardness, GX_0-Real reduces to GX-Real following Lemma 13 above: every GX_0 formula can be efficiently translated into an equirealizable GX formula.
\end{proof}

It seems to us that GX_0 is a particularly interesting fragment for realizability. On the one hand, it is provably easier that full LTL. On the other, its specifications are all safety properties of a very natural flavor: we specify what the system should to at the beginning and then enforce that it always reacts appropriately. This general pattern is expressive enough to capture most industrial cases, in a way similar to GR(1) formulas (a fragment whose realizability problem is also EXP-complete [10]). The advantage, however, is that GX_0 specifications are conceptually simpler than GR(1) ones, and they can be simplified even further, by removing the initial condition and getting a GX specification. Furthermore, as our next theorem shows, GX_0 suffices to capture the full power of SafetyLTL modulo equirealizability.


4 Realizability for Safety LTL

The central fragment of LTL when it comes to realizability is Safety LTL (see Definition 1). This corresponds to the Until-free fragment of LTL in NNF. Since Safety LTL can express all the safety properties expressible in LTL (see, for example, [15]), being able to solve instances of realizability in this language is of central importance.

We now show that solving realizability instances written in the language of Safety LTL amounts to solving an equirealizable instance from GX or GX0. In order words, the realizability problem for Safety LTL has the same complexity as for the other two fragments, even despite having seemingly more powerful operators like Release ($R$). Surely, Safety LTL and GX0 are not expressively equivalent in the strict sense, but using a few auxiliary atoms suffices to simulate the desired operators. That is, $GX \equiv \text{Safety LTL}$.

\textbf{Theorem 15.} The realizability problem for Safety LTL is EXP-complete.

\textbf{Proof.} In this case, hardness is the simpler part. It follows from the observation that $\Box \varphi$ can be written as $\perp R \varphi$, meaning that a formula $\alpha \land \Box \psi$ of $GX_0$ is already a Safety LTL formula, and hence $GX_0$-REAL, which is EXP-complete (Corollary 12), reduces to this problem, giving us EXP-hardness for Safety LTL realizability.

To show membership, we give the reverse reduction: given a formula $\varphi$ of Safety LTL, we provide an equirealizable $GX_0$ formula. We do it by induction on the structure of $\varphi$.

- If $\varphi = p$, take a new system variable $s$ and consider the $GX_0$ specification $\alpha \land \Box \psi$ defined by $\alpha := s$ and $$\psi := (s \rightarrow (p \land \Diamond \neg s)) \land (\neg s \rightarrow \Diamond \neg s).$$ Note how if $p$ is an environment variable, then $\varphi = p$ is trivially unrealizable, and so is the translated version. If $p$ is a system variable, the new version is also trivially realizable. The case for $\varphi = \neg \varphi$ is analogous, and since Safety LTL formulas are in Negation Normal Form, this takes care of all occurrences of negation.

- If $\varphi = \varphi_1 \land \varphi_2$, then by induction hypothesis $\varphi_1 \equiv \alpha_1 \land \Box \psi_1 \in GX_0$ and $\varphi_2 \equiv \alpha_2 \land \Box \psi_2 \in GX_0$, and we can easily put them together: $(\alpha_1 \land \alpha_2) \land (\Box \psi_1 \land \Box \psi_2) = \varphi$, up to renaming of auxiliary variables.

- If $\varphi = \varphi_1 \lor \varphi_2$, then by induction hypothesis $\varphi_1 \equiv \alpha_1 \land \Box \psi_1 \in GX_0$ and $\varphi_2 \equiv \alpha_2 \land \Box \psi_2 \in GX_0$, and we can easily put them together with an auxiliary system variable $c$,

$$\alpha := (c \rightarrow \alpha_1) \land (\neg c \rightarrow \alpha_2) \quad \psi := (c \rightarrow (\psi_1 \land \Diamond c)) \land (\neg c \rightarrow (\psi_2 \land \Diamond \neg c))$$

where $c$ acts as a “choice” variable. Note how the equirealizability of the induction hypothesis gives us $\varphi_1 \lor \varphi_2 = \alpha \land \Box \psi$.

- If $\varphi = \Diamond \mu$, then by induction hypothesis $\mu = \alpha \land \Box \psi \in GX_0$. We introduce a new auxiliary system variable $s$ to indicate that we are at the first time step, getting the formulas

$$\beta := s \quad \chi := ((s \rightarrow \Diamond \alpha \land \Diamond \neg s) \land (\neg s \rightarrow \psi \land \Diamond \neg s))$$

which gives a specification $\beta \land \Box \chi$ equirealizable to $\varphi$.

- If $\varphi = \varphi_2 R \varphi_1$, then, by induction hypothesis, $\varphi_1$ is equirealizable to some $\alpha_1 \land \Box \psi_1 \in GX_0$, and $\varphi_2$ is equirealizable to some $\alpha_2 \land \Box \psi_2 \in GX_0$. By Lemma 13, every $GX_0$ formula can be efficiently converted into an equirealizable $GX$ formula. Hence, there are formulas $\Box \chi_1 \in GX$ and $\Box \chi_2 \in GX$ equirealizable to $\alpha_1 \land \Box \psi_1$ and $\alpha_2 \land \Box \psi_2$, respectively. Then, $\varphi_2 R \varphi_1$ is realizable if and only if $\Box \chi_2 R \Box \chi_1$ is realizable. By the semantics of $R$, every model that makes $\Box \chi_1$ true will also make $\Box \chi_2 R \Box \chi_1$ true, and vice versa, so $\varphi_2 R \varphi_1$ is realizable if and only if $\Box \chi_1$ is realizable, which is a $GX \subseteq GX_0$ formula, as desired.
This completes the induction: every Safety LTL formula can be efficiently converted to a GX₀ formula, and hence realizability for Safety LTL reduces to realizability for GX₀, which is EXP-complete, giving us the desired membership of Safety LTL realizability in EXP.

4.1 Comparision with existing expressibility results

Since realizability for both GX₀ and Safety LTL is EXP-complete, this means that GX₀ = Safety LTL (they are equirealizable fragments). Observe how this is in contrast with the fact that GX₀ is a strict subfragment of Safety LTL, both syntactically (GX₀ ⊆ Safety LTL) and semantically ([GX₀] ⊆ [Safety LTL]). The latter follows from one of the expressibility results of Cimatti et al. in [7]. Let us say something about these results in comparison to ours.

Cimatti et al. work with two fragments of LTL, called LTL_{EBR} and LTL_{EBR} + P, where the latter is an extension of the former by adding past operators. A formula ψ of LTL_{EBR} + P is built according to the following grammar:

\[
\begin{align*}
\pi ::= p | \neg \pi | \pi \lor \pi | \Box \pi | \Box S \pi & \quad \text{(Past Layer)} \\
\beta ::= p | \neg \beta | \beta \lor \beta | \Box \beta | \beta U_{(a,b)} \beta & \quad \text{(Bounded Future Layer)} \\
\phi ::= \beta | \phi \land \phi | \Box \phi | \Box D \phi & \quad \text{(Future Layer)} \\
\psi ::= \phi | \psi \lor \psi | \psi \land \psi & \quad \text{(Boolean Layer)}
\end{align*}
\]

where the bounded Until operator \( \beta U_{(a,b)} \beta \), with \( a, b \in \mathbb{N} \), \( a \leq b \), is a shortcut for the formula

\[
\bigvee_{i=a}^{b} \left( \Box^{i} \beta_{2} \land \bigwedge_{j=0}^{i-1} \Box^{j} \beta_{1} \right).
\]

Then, the fragment LTL_{EBR} is obtained from the previous grammar, by removing the Past Layer.

Observe that, syntactically, GX₀ ⊆ LTL_{EBR} ⊆ Safety LTL. Note that LTL_{EBR} is almost syntactically equal to Safety LTL, but not fully: in the Future Layer of LTL_{EBR}, Release (R) cannot have universal temporal operators nested in the left-hand-side, while this is allowed in Safety LTL. For example, \((\Box p) R q \in Safety LTL \setminus LTL_{EBR}\).

Semantically, it is also the case that LTL_{EBR} is strictly less expressive than Safety LTL, but adding the Past Layer suffices to bridge that gap. They show that \([LTL_{EBR} + P] = [[\Box \pi \mid \pi \in P]]\). That is, the formulas in LTL_{EBR} + P are expressively equivalent to formulas of the form \( \Box \pi \), where \( \pi \) is a pure past formula. It turns out that the fragment of formulas \( \Box \pi \) captures all the safety properties expressible in LTL (see [3]), just like Safety LTL. Hence, \([LTL_{EBR} + P] = [Safety LTL]\). It is important to underline, however, that despite both fragments having the same expressive power, there is no known explicit translation between them. In fact, since LTL_{EBR} + P contains past modalities, some of its formulas might be exponentially more succinct than the equivalent Safety LTL ones, so the translation in that direction might lead to a blow-up in the size of the formulas.

Furthermore, they argue that the past layer of LTL_{EBR} + P is really necessary by showing that \([LTL_{EBR}] \not\subseteq [Safety LTL]\). To prove this, they show that the formula \( \Box (p \lor q) \in Safety LTL \not\subseteq [Safety LTL] \) is not expressible in LTL_{EBR}.

---

1 Technically, this is LTL_{EBR} without constants. If \( a \) and \( b \) are represented in binary in the input, this leads to an exponential blow-up that raises complexity from EXP to 2EXP, so in order to remain EXP-complete, \( \beta U_{(a,b)} \beta \) takes \( a \) and \( b \) in unary, which is as concise as writing the formula it stands for.
Fortunately, since Safety LTL is equirealizable to $\text{GX}_0$, we can translate their counter-example formula into an equirealizable $\text{GX}_0$ formula, as before. The formula $\square(p \lor \square q)$ has two types of models: either $p$ is always true or $p$ is true for a finite amount of time, after which $q$ is true forever. We introduce a system variable $s$ indicating that we are in the “simple” model, the one where $p$ is always true. Then, we consider the $\text{GX}_0$ specification

$$\alpha := s$$

$$\psi := (s \land p \rightarrow \bigcirc s) \land (s \land \neg p \rightarrow q \land \bigcirc \neg s) \land (\neg s \rightarrow q \land \bigcirc \neg s)$$

which gives us $\alpha \land \square \psi = \square (p \lor \square q)$. Observe that the equirealizability holds regardless of whether $p$ and $q$ belong to the system or to the environment. If at least one of the two variables belongs to the system, then $\square (p \lor \square q)$ is realizable, and so is $\alpha \land \square \psi$. If both $p$ and $q$ belong to the environment, then both specifications are unrealizable. Hence, even if $[\text{GX}_0] \subseteq [\text{Safety LTL}]$, as long as we are only interested in realizability, all the Safety LTL specifications can be easily translated into the simpler $\text{GX}_0$ introducing auxiliary atoms, as in the proof of Theorem 15.

### 5 Conclusion and future work

We have approached the realizability problem for Safety LTL from a purely complexity-theoretic perspective. We have proven that the problem is EXP-complete (Theorem 15), improving the double-exponential complexity of the best existing algorithms, by showing that Safety LTL is equirealizable to the new fragment $\text{GX}_0$ that we introduce. Surprisingly, the simple technique of adding auxiliary atoms in the translations had not yet been explored for realizability problems.

This proof is more than a theoretical curiosity: our reductions present direct and efficient translations between equirealizable fragments, which open the door to a simpler approach to the design of realizability algorithms. Instead of designing highly specialized languages and intricate algorithms for them, we suggest coming up with algorithms for the simple $\text{GX}_0$ fragment, which can solve equirealizability for Safety LTL via our translation. Furthermore, as long as the algorithm employed is able to extract a winning strategy, the translation can be reversed, extracting a winning strategy for the original input formula. That is, an algorithm using this translation would be able to solve synthesis on top of realizability. We are already working on such an algorithm, together with a tableau-based proof system for $\text{GX}_0$ realizability. We expect to publish this in the near future.

The idea of reducing hard problems to more intuitive ones of the same structural complexity is not at all new in complexity theory, of course. This has traditionally been the main approach to attack NP problems in practice, via reductions to SAT and calls to highly optimized satisfiability algorithms. The success of that approach in the realm of NP problems leads us to think that this same line of research can be fruitful for the realizability and synthesis of reactive systems.

More generally, we have shown that simple complexity-theoretic techniques can still be of great use and open new lines of investigation. Structural complexity is hence an additional lens that can be used to compare fragments of logics designed for specific problems, in a more fundamental way than traditional expressibility and logical equivalence.

### Acknowledgements

We would like to thank the user DCTLib from Computer Science Stack Exchange, who in [9] informally sketched the proof of Corollary 12 and suggested the idea of a “resetting variable”
used in the proof of Lemma 13.

We also thank César Sánchez, from the IMDEA Software Institute in Madrid, Spain, for useful references and suggestions.

References

1. Rajeev Alur and Salvatore La Torre. Deterministic generators and games for LTL fragments. *ACM Transactions on Computational Logic (TOCL)*, 5(1):1–25, 2004.
2. Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
3. Edward Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In *International Colloquium on Automata, Languages, and Programming*, pages 474–486. Springer, 1992.
4. Chih-Hong Cheng, Yassine Hamza, and Harald Ruess. Structural synthesis for GXW specifications. In *International Conference on Computer Aided Verification*, pages 95–117. Springer, 2016.
5. Alonzo Church. Logic, arithmetic, and automata. *Journal of Symbolic Logic*, 29(4), 1964.
6. Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. Reactive synthesis from Extended Bounded Response ltl specifications. In *2020 Formal Methods in Computer Aided Design (FMCAD)*, pages 83–92. IEEE, 2020.
7. Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. Expressiveness of extended bounded response LTL. *arXiv preprint arXiv:2109.08319*, 2021.
8. Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. Extended Bounded Response LTL: A new safety fragment for efficient reactive synthesis. *Formal Methods in System Design*, pages 1–49, 2021.
9. DCTLib. Complexity of LTL realizability of safety games with Next operator only. Computer Science Stack Exchange. (version: 2021-08-07). [URL](https://cs.stackexchange.com/q/142988).
10. Rüdiger Ehlers. Symmetric and efficient synthesis. 2013.
11. Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.
12. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
13. Nir Piterman, Amir Pnueli, and Yaniv Sa’ar. Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.
14. Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In *International Colloquium on Automata, Languages, and Programming*, pages 652–671. Springer, 1989.
15. Salomon Sickert and Javier Esparza. An efficient normalisation procedure for linear temporal logic and very weak alternating automata. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 831–844, 2020.
16. A Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994.
17. Shufang Zhu, Lucas M Tabajara, Jianwen Li, Geggau Pu, and Moshe Y Vardi. A symbolic approach to Safety LTL synthesis. In *Hardware and Software: Verification and Testing: 13th International Haifa Verification Conference, HVC 2017, Haifa, Israel, November 13-15, 2017, Proceedings*, volume 10629, page 147. Springer, 2017.
A Converting GXW specifications into GX₀ specifications

The complexity results proven entail that one can often convert formulas in a given language to equirealizable formulas of a related fragment. A particularly interesting fragment is GXW, capable of capturing a wide range of industrial specifications while still keeping a low complexity, since its realizability problem is in PSPACE [4]. In Section 3.1 we briefly discussed that GXW realizability being in PSPACE implies a translation into X (or QBF) formulas. As a different example, we now give instead a conversion to the GX₀ fragment in order to display its expressive power modulo equirealizability.

In what follows, we denote by ϕᵢₜₜ a formula over system (respectively environment) variables where the only temporal operator is Next and the maximum number of nested occurrences of it (i.e., its temporal depth) is i. In particular, ϕ₀ₜₜ and ϕ₀ₜ are present formulas over system or environment variables. In addition, we use ℓₜₜ and ℓₜ to refer to system or environment literals, respectively.

A GXW specification is a formula of the form

□ϕ₀ₜₜ → ⋀ₖ=₁ₘ ηₖ

where each ηₖ adheres to one of the following five patterns:

(P₁) ℓₜₜWϕᵢₜₜ
(P₂) □(ϕᵢₜₜ → □(ℓₜₜW(ϕᵢₜₜ ∨ χ₀ₜₜ)))
(P₃) □(ϕᵢₜₜ → □ℓₜₜ)
(P₄) □(ϕᵢₜₜ ↔ □ℓₜₜ)
(P₅) □ϕ₀ₜ

Let □ϕ₀ₜₜ → ⋀ₖ=₁ₘ ηₖ be a GXW specification. The idea for the translation is to use auxiliary system variables to emulate operators like W, which are available in GXW but not in GX₀. Roughly, our procedure goes as follows. First, we show how to translate every pattern into a GX₀ specification. Then, taking the conjunction and doing some minor modifications, we will easily get a formula α∧ψ that is equirealizable to the consequent of the GXW formula. Finally, we describe how to combine the antecedent with α∧ψ to get a GX specification β∧ψ.

Note that patterns (P₃)-(P₅) are already GX specifications, so we only need to translate (P₁) and (P₂).

Translation of the (P₁) pattern. Let ℓₜₜWϕᵢₜₜ be a formula satisfying the pattern. Recall the semantics of the weak until operator: ℓₜₜ remains true until ϕᵢₜₜ becomes true, but ϕᵢₜₜ is not forced to eventually do so. Then, we introduce an auxiliary system variable s indicating that ϕᵢₜₜ has not become true yet. Then, consider the following specification α∧□ψ, where

α := s
ψ := (s ∧ ¬ϕᵢₜₜ → ℓₜₜ ∧ □s) ∧ (s ∧ ϕᵢₜₜ → □¬s) ∧ (¬s → □¬s)

Observe that ℓₜₜWϕᵢₜₜ is realizable if and only if α∧□ψ is realizable.

Translation of the (P₂) pattern. The idea here is again the same, though the translation is arguably more complicated. Consider a formula

□(ϕᵢₜₜ → □(ℓₜₜW(ϕᵢₜₜ ∨ χ₀ₜₜ)))
in the shape of the pattern. The subformula \( \ell_{\text{out}} W(\psi_{\text{in}} \vee \chi_{\text{out}}^0) \) can be translated into an equirealizable \( \text{GX}_0 \) formula \( \alpha \land \Box \psi \) by using the same trick as for the first pattern. Then, the pattern becomes equirealizable to

\[
\Box (\varphi_{\text{in}}^i \rightarrow \Box^i (\alpha \land \Box \psi))
\]

which we can translate into \( \text{GX}_0 \) by introducing an auxiliary system variable \( h \), keeping track of whether \( \varphi_{\text{in}}^i \) has ever become true. Then, consider the formulas

\[
\beta := \neg h
\]

\[
\chi := (\varphi_{\text{in}}^i \rightarrow \Box^i \alpha) \land (\varphi_{\text{in}}^i \land \neg h \rightarrow \Box^i (h \land \psi)) \land (h \rightarrow \psi \land \Box h)
\]

Then, observe that the specification \( \beta \land \Box \chi \in \text{GX}_0 \) is equirealizable to the original pattern.

Hence, for every \( \eta_k \), we have an equirealizable formula \( \alpha_k \land \Box \psi_k \). Therefore, letting \( \alpha := \bigwedge_{k=1}^{m} \alpha_k \) and \( \psi := \bigwedge_{k=1}^{m} \psi_k \), we have that \( \bigwedge_{k=1}^{m} \eta_k \) is equirealizable to \( \alpha \land \Box \psi \), which is a \( \text{GX}_0 \) specification.

It now remains to combine the antecedent \( \Box \varphi_{\text{in}}^0 \) of the \( \text{GXW} \) specification together with the formula \( \alpha \land \Box \psi \) into an equirealizable formula \( \beta \land \Box \chi \). We cannot immediately do this, because \( \Box \) does not distribute over disjunction, but we can simulate this behavior using an auxiliary variable. Intuitively, the antecedent \( \Box \varphi_{\text{in}}^0 \) ensures that, at all times, the environment chooses a legal assignment. We cannot impose this condition in the same way, but since the specification \( \psi \) that has to be satisfied has finite temporal depth \( d \), it actually suffices to check, at every point in time and before enforcing the realizability of \( \psi \), that the environment makes legal assignments in the interval \([0,d]\). With this in mind, we consider an auxiliary variable \( l \) to indicate that the current assignment is legal:

\[
\beta := \alpha \land l
\]

\[
\chi := (\neg l \rightarrow \Box \neg l) \land (l \rightarrow ((\Box [0,d] \varphi_{\text{in}}^0 \land \psi \land \Box l) \lor (\Box [0,d] \neg \varphi_{\text{in}}^0 \land \Box \neg l)))
\]

Hence, \( \beta \land \Box \chi \) is equirealizable to the original \( \text{GXW} \) specification. Furthermore, the translation process is efficient. Hence, we have an example of the reduction from \( \text{GXW} \) realizability to \( \text{GX}_0 \) realizability, as desired.

**B** Formulas in the proof of Proposition 11

We explicitly write the formulas in the specification \( \alpha \land \Box \psi \) of Proposition 11.

Using the variables defined, one first consider a Boolean formula \( \alpha \), expressing that at the beginning, the machine is at state \( q_0 \) and the tape in empty:

\[
\alpha := p_{1,q_0} \land \bigwedge_{i \in [s]} c_i
\]

Then, consider the auxiliary formulas \( \mathcal{E} \) and \( \mathcal{U} \), expressing that the current state is existential (respectively universal) and inside the space bounds:

\[
\mathcal{E} := \bigvee_{i \in [s], q \in Q_3} p_{i,q} \quad \mathcal{U} := \bigvee_{i \in [s], q \in Q_4} p_{i,q}
\]

Now, we can write a temporal formula \( \psi \) where \( \Box \) is the only temporal operator expressing the following constraints:
(S) The machine must be in one and only one state, reading one and only one cell at a time:

\[ \bigvee_{i \in [s], q \in Q} \left( p_{i,q} \land \bigwedge_{j \in [s] \setminus \{i\}, q' \in Q \setminus \{q\}} \neg p_{j,q'} \right) \]

(I) If the current state is illegal (i.e., either the rejecting state or above the space limits), then keep the state of the machine unchanged for the next state:

\[ \bigwedge_{i \in [s]} \left( p_{i,q_{\text{reject}}} \rightarrow O p_{i,q_{\text{reject}}} \right) \land \bigwedge_{q \in Q} \left( p_{s+1,q} \rightarrow O p_{s+1,q} \right) \]

(T) At existential states, the system must choose a transition function (\( \delta \) or \( \delta' \)) and the state of the machine must be updated accordingly for the next state. For instance, consider the case when the state is existential and the system chooses \( \delta \). We need to consider four different cases for how to update everything for the next state, depending on the symbol being read and the symbol being written on the tape. That is,

\[ \mathcal{E} \land \delta \rightarrow \bigvee_{i \in [s], q \in Q} \left( p_{i,q} \land c_i \land O p_{i+d,q'} \land O c_i \land \bigwedge_{j \neq i} c_j \leftrightarrow O c_j \right) \]

\[ \bigvee_{i \in [s], q \in Q} \left( p_{i,q} \land c_i \land O p_{i+d,q'} \land O \neg c_i \land \bigwedge_{j \neq i} c_j \leftrightarrow O c_j \right) \]

\[ \bigvee_{i \in [s], q \in Q} \left( p_{i,q} \land \neg c_i \land O p_{i+d,q'} \land O c_i \land \bigwedge_{j \neq i} c_j \leftrightarrow O c_j \right) \]

\[ \bigvee_{i \in [s], q \in Q} \left( p_{i,q} \land \neg c_i \land O p_{i+d,q'} \land O \neg c_i \land \bigwedge_{j \neq i} c_j \leftrightarrow O c_j \right) \]

Similarly, there will be analogous implications for \( \mathcal{E} \land \neg \delta \rightarrow \ldots, U \land \delta \rightarrow \ldots \) and \( U \land \neg \delta \rightarrow \ldots \) to cover all possible cases.

(F) Finally, impose that the accepting state is never visited:

\[ \bigwedge_{i \in [s]} \neg p_{i,q_{\text{accept}}} \]

Now, let \( \psi := \mathcal{S} \land \mathcal{I} \land \mathcal{T} \land \mathcal{F} \). This completes the specification \( \alpha \land \square \psi \).