Fairness, Assumptions, and Guarantees for Extended Bounded Response LTL+P synthesis

Alessandro Cimatti[1][0000−0002−1315−6990], Luca Geatti[1,2][0000−0002−7125−787X], Nicola Gigante[3][0000−0002−2254−4821], Angelo Montanari[2][0000−0002−4322−769X], and Stefano Tonetta[1][0000−0001−9091−7899]

[1] Fondazione Bruno Kessler, Trento, Italy
{cimatti,lgeatti,tonettas}@fbk.eu
[2] University of Udine, Italy
{luca.geatti,angelo.montanari}@uniud.it
[3] Free University of Bozen-Bolzano, Italy
nicola.gigante@unibz.it

Abstract. Realizability and reactive synthesis from temporal logics are fundamental problems in the formal verification field. The complexity of the Linear-time Temporal Logic with Past (LTL+P) case led to the definition of fragments with lower complexities and simpler algorithms. Recently, the logic of Extended Bounded Response LTL+P ($\mathsf{LTL}_{\mathsf{EBR}}$+P) has been introduced. It allows one to express any safety language definable in LTL and it is provided with an efficient, fully-symbolic algorithm for reactive synthesis.
In this paper, we extend $\mathsf{LTL}_{\mathsf{EBR}}$+P with fairness conditions, assumptions, and guarantees. The resulting logic, called GR-EBR, preserves the main strength of $\mathsf{LTL}_{\mathsf{EBR}}$+P, that is, efficient realizability, and makes it possible to specify properties beyond safety. We study the problem of reactive synthesis for GR-EBR and devise a fully-symbolic algorithm that reduces it to a number of safety subproblems. To ensure soundness and completeness, we propose a general framework for safety reductions in the context of realizability of (fragments of) LTL+P. The experimental evaluation shows the feasibility of the approach.

Keywords: Realizability · Temporal Logic · Symbolic Automata

1 Introduction

One of the most important problems in formal methods and requirement analysis is establishing whether a specification over a set of controllable and uncontrollable actions is *implementable* (or *realizable*), that is, whether there exists a controller that chooses the value of the controllable actions and satisfies the specification, no matter what the values of uncontrollable actions are. This problem has been formalized in the literature under the name of *realizability* [3]. The very close problem of *reactive synthesis* aims at synthesizing such a controller, whenever the specification is realizable. Usually, these problems are modelled as

two-player games between Environment, who tries to violate the specification, and Controller, who tries to fulfill it. Realizability is known to have a very high worst-case complexity. In particular, it has a non-elementary lower bound for S1S specifications [2], and it is 2EXPTIME-complete for LTL specifications [22, 23].

In order to apply realizability and reactive synthesis in real-world scenarios, research has focused on the identification of fragments of logics like S1S and LTL, with a limited expressive power, for which realizability can be solved efficiently.

A well-known example is *Generalized Reactivity(1)* logic (GR(1), for short) [1]. In this fragment, a specification is syntactically partitioned into *assumptions* about the environment and *guarantees* for the controller. Both of them are either Boolean formulas (α) or safety formulas ($\mathsf{G}\alpha$) or conjunctions of recurrence formulas ($\bigwedge_{i=1}^{n} \mathsf{GF}\alpha_i$). The dichotomy between *assumptions* and *guarantees* reflects the way a system engineer usually formalizes system's requirements, which is summarized by the following sentence: *"the controller has to behave in conformance to the guarantees, under the given assumptions on the environment"*.

On a different direction, other approaches focused on safety fragments of LTL [4, 25]. In particular, *Extended Bounded Response LTL* ($\mathsf{LTL}_{\mathsf{EBR}}$, for short) is a safety fragment of LTL+P that allows for a fully symbolic compilation of formulas into deterministic automata. Such a feature contributes to a great improvement in solving time for the synthesis problem. Moreover, $\mathsf{LTL}_{\mathsf{EBR}}$ has a well-established expressiveness: $\mathsf{LTL}_{\mathsf{EBR}}$ can define exactly the set of safety languages definable in LTL.

Contributions The main contributions of this paper are the following ones. First, we introduce GR-EBR, an extension of $\mathsf{LTL}_{\mathsf{EBR}}$ that admits: (i) fairness conditions, in particular, conjunctions of *recurrence formulas*, that is, $\bigwedge_i \mathsf{GF}\alpha_i$, forcing each formula α_i to be true infinitely often; (ii) assumptions/guarantees in the form of an $\mathsf{LTL}_{\mathsf{EBR}}$ formula augmented with fairness conditions. In addition to be able to express any $\mathsf{LTL}_{\mathsf{EBR}}$ formula, and, consequently, any safety property definable in LTL, GR-EBR allows also for the definition of properties beyond the safety fragment, like, for instance, $\mathsf{G}(p) \rightarrow \mathsf{G}(q)$.

Second, we devise a novel framework for deriving *complete* safety reductions in the context of realizability of (fragments of) LTL. A notable feature of the framework is that it provides a link to safety reductions for the model checking problem and proves that if a reduction is complete for model checking, then it is also complete for realizability. On one hand, this allows one to reason on Kripke structures instead of on strategies, which is simpler; on the other hand, it enables the use of some reductions already exploited in model checking for realizability, provided that they conform to the framework.

Third, the proposed framework is used to derive a *complete* safety reduction for the realizability problem of GR-EBR. A crucial property of the algorithm is that the realizability check of each safety sub-problem is performed in a *fully symbolic* way, thus retaining the distinctive feature of $\mathsf{LTL}_{\mathsf{EBR}}$.

Last but not least, we provide an implementation of the algorithm as a prototype tool called GRACE (*GR-ebr reAlizability ChEcker*). The experimental evaluation shows good performance against tools for full LTL+P synthesis.

Related work GR(1) has been introduced in [1, 21]. It is known that GR(1) is a good candidate for writing specifications of real-world scenarios, with a relatively low complexity: the realizability problem can be solved with at most a quadratic number of symbolic steps in the size of the specification [1]. On the other hand, GR(1) presents some restrictions that limit its use as a specification language: (i) safety assumptions/guarantees are either Boolean formulas or formulas of the form $\mathsf{G}\alpha$, where the only temporal operator admitted in α is the *next* operator X; (ii) assumptions are syntactically constrained to be formulas *controlled* by Environment, in the sense that the variables inside the *next* operators of the safety part of the assumptions must be *uncontrollable*. In GR-EBR we relax that syntactical restrictions of GR(1): for example, the safety assumptions and guarantees can be any arbitrary $\mathsf{LTL}_{\mathsf{EBR}}$ formula, like, for instance, $\mathsf{G}(r \rightarrow \mathsf{F}^{[0,10]}g)$. For this reason, GR-EBR can be considered an extension not only of $\mathsf{LTL}_{\mathsf{EBR}}$, but also of GR(1). On the semantic side, the comparison is more problematic. On the one hand, all (standard) realizability problems for GR(1) specifications are definable in LTL+P [1] and also in GR-EBR. On the other hand, we do not known whether GR-EBR is able to express more properties than GR(1). Our conjecture is that this is the case. Take for instance the bounded-response property $\mathsf{G}(r \rightarrow \mathsf{F}^{[0,k]}g)$: it is easily expressible in GR-EBR, but we see no way it could be definable in GR(1) without introducing additional variables (that would maintain realizability but not language equivalence).

Bounded synthesis [9, 13] belongs to the class of *Safraless techniques* [17], and it consists in bounding the number of times Controller is forced to visit a rejecting state of a Universal co-Büchi automaton (UCW, for short) for the initial formula. This corresponds to a safety automaton, which can be either (i) made deterministic by a suitable generalization of the classical subset construction [7, 11], or (ii) encoded into a constraint system [9, 13] (*e.g.*, SAT- or SMT-based) which bounds also the *size* of a candidate controller (this also allows one to tackle undecidable problems, for instance in the case of distributed or parametric synthesis). Both choices work for the whole class of UCW, and thus for full LTL. A significant drawback of such an approach is that the UCW, which can be exponentially larger than the initial specification, is explicitly represented. Moreover, in the first case, the algorithm for the determinization turns out to be quite complex, since each state of the resulting automaton is actually a *function*. This can also result into a very large state space, that can be tackled by exploiting either *antichains* [11] or BDDs [7]. In contrast, as we will see, we define a reduction tailored to GR-EBR formulas that allows us to exploit the $\mathsf{LTL}_{\mathsf{EBR}}$ transformations introduced in [4] for a *fully symbolic* mapping of the initial formula directly into a sequence of symbolic safety automata. In particular, we never build any explicit-state automaton and we avoid the subsequent use of determinization algorithms.

Organization The rest of the paper is organized as follows. In Sec. 2, we introduce the notation and provide the basic definitions. In Sec. 3, we define the logic GR-EBR and give an example of GR-EBR specification. The framework for deriving complete reductions is presented in Sec. 4. In Sec. 5 we describe the

algorithm for the realizability of GR-EBR specifications. The outcomes of the experimental evaluation are reported in Sec. 6. Finally, in Sec. 7, we point out some interesting future research directions.

2 Preliminaries

2.1 Temporal Logics

Linear Temporal Logic with Past (LTL+P) is a modal logic interpreted over infinite state sequences. Let Σ be a set of propositions. LTL+P formulas are inductively defined as follows:

$$\phi := p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{X}\phi \mid \phi_1 \,\mathsf{U}\, \phi_2 \mid \mathsf{Y}\phi \mid \phi_1 \,\mathsf{S}\, \phi_2$$

where $p \in \Sigma$. Temporal operators can be subdivided into the *future operators*, *next* (X) and *until* (U), and *past operators*, *yesterday* (Y) and *since* (S). We define the following common abbreviations (where $\top$ stands for any tautology such as $p \vee \neg p$): (i) *release*: $\phi_1 \,\mathsf{R}\, \phi_2 \equiv \neg(\neg\phi_1 \,\mathsf{U}\, \neg\phi_2)$; (ii) *eventually*: $\mathsf{F}\phi_1 \equiv \top \,\mathsf{U}\, \phi_1$; (iii) *globally*: $\mathsf{G}\phi_1 \equiv \neg\mathsf{F}\neg\phi_1$; (iv) *once*: $\mathsf{O}\phi_1 \equiv \top \,\mathsf{S}\, \phi_1$; (v) *historically*: $\mathsf{H}\phi_1 \equiv \neg\mathsf{O}\neg\phi_1$. LTL+P formulas are interpreted over infinite state sequences (or ω-words) $\pi \in (2^{\Sigma})^{\omega}$. We call *language* a set of ω-words. We write $\pi \models \phi$ to denote the fact that the state sequence π is a *model* (or *satisfies*) the formula ϕ. We refer to [4] for the semantics of the LTL+P operators. With $|\phi|$, we refer to the *size* of the formula ϕ, defined as the number of symbols in it. We define the *language* of ϕ, written $\mathcal{L}(\phi)$, as the set of all and only the models of ϕ.

An important class of languages is the class of safety properties, that express the fact that "something bad never happens". Formally, we define a *safety property* (or *safety language*) as a language for which it holds that, for any ω-word that does not belong to language, there exists a finite prefix of it such that all its continuations do not belong to the language as well. A formula ϕ is called a *safety formula* if $\mathcal{L}(\phi)$ is a safety language. Recently, Cimatti *et al.* [4] introduced a subset of LTL+P, called *Extended Bounded Response* LTL+P, which expresses exactly the safety properties that can be defined in LTL+P [5], and gave a symbolic procedure to turn formulas of this fragment into symbolic automata.

Definition 1 (The logic $\mathsf{LTL}_{\mathsf{EBR}}$+P [4]). *Let $a, b \in \mathbb{N}$. An $\mathsf{LTL}_{\mathsf{EBR}}$+P formula χ is inductively defined as follows:*

$$\eta := p \mid \neg\eta \mid \eta_1 \vee \eta_2 \mid \mathsf{Y}\eta \mid \eta_1 \,\mathsf{S}\, \eta_2 \qquad \textit{Pure Past Layer}$$

$$\psi := \eta \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{X}\psi \mid \psi_1 \,\mathsf{U}^{[a,b]}\, \psi_2 \qquad \textit{Bounded Future Layer}$$

$$\phi := \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid \psi \,\mathsf{R}\, \phi \qquad \textit{Future Layer}$$

$$\chi := \phi \mid \chi_1 \vee \chi_2 \mid \chi_1 \wedge \chi_2 \qquad \textit{Boolean Layer}$$

We define the *bounded until* operator $\psi_1 \,\mathsf{U}^{[a,b]}\, \psi_2$ as a *shortcut* of the formula $\bigvee_{i=a}^{b}(\mathsf{X}^i\psi_2 \wedge \bigwedge_{j=0}^{i-1}\mathsf{X}^j\psi_1)$, where $\mathsf{X}^i\phi := \mathsf{X}_{(1)}\ldots\mathsf{X}_{(i)}\phi$. We define $\mathsf{LTL}_{\mathsf{P}}$ (the *pure past fragment of LTL+P*) as the set of all the formulas belonging to the *Pure*

Past Layer of Def. 1, respectively. With some abuse of notation, we will denote with the symbol of the logic (*e.g.*, $\mathsf{LTL}{+}\mathsf{P}$ or $\mathsf{LTL}_{\mathsf{EBR}}{+}\mathsf{P}$) also the *set* of all the formulas of the respective logic.

2.2 Automata

Temporal logic has a strong relation with automata on infinite words [24]. Since in the following we will work only with symbolic representations, we give here the definition of symbolic automata. It is well-known that the symbolic representation can be exponentially more succinct than the explicit-state one.

Definition 2 (Symbolic Automaton on Infinite Words). *A* symbolic automaton on infinite words *over the alphabet Σ is a tuple $\mathcal{A} = (V, I, T, \alpha)$, such that (i) $V = X \cup \Sigma$, where X is a set of* state variables *and Σ is a set of* input variables*, (ii) $I(X)$ and $T(X, \Sigma, X')$, with $X' = \{x' \mid x \in X\}$, are Boolean formulas which define the set of initial states and the transition relation, respectively, and (iii) $\alpha(X)$ is an $\mathsf{LTL}{+}\mathsf{P}$ formula over the variables in X which defines the accepting condition.*

Definition 3 (Languages of Symbolic Automata). *An ω-word (or simply a word) $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ is an infinite sequence of letters in Σ. A run $\tau = \langle \tau_0, \tau_1, \ldots \rangle$ is an infinite sequence of* states *(*i.e.*, evaluations of the variables in X) that are in relation with respect to T (*i.e.*, such that any two consecutive evaluations satisfy the formula T). A run τ is induced by the word σ iff $\tau_0 \models I$ and $(\tau_i, \sigma_i, \tau_{i+1}) \models T$, for all $i \geq 0$. We say that $\mathcal{A}$ is* deterministic *iff there exists exactly one trace induced by σ, for each $\sigma \in \Sigma^\omega$. A word σ is* accepted *by $\mathcal{A}$ iff there exists an accepting run induced by σ in $\mathcal{A}$. The* language *of $\mathcal{A}$, denoted with $\mathcal{L}(\mathcal{A})$, is the set of all and only the words accepted by $\mathcal{A}$.*

We will refer to three important classes of accepting conditions: (i) *safety*: $\alpha(X) \coloneqq \mathsf{G}\beta$; (ii) *Reactivity(1)*: $\alpha(X) \coloneqq \mathsf{GF}\beta \rightarrow \mathsf{GF}\beta'$; (iii) *Generalized Reactivity(1)*: $\alpha(X) \coloneqq \bigwedge_{i=1}^{m} \mathsf{GF}\beta_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta'_j$; where each $\beta, \beta', \beta_i, \beta'_j \in \mathsf{LTL}_{\mathsf{P}}$.

2.3 Model Checking, Realizability, and Synthesis

A Kripke structure is a tuple $M = (\Sigma, Q, I, T, L)$ where: (i) Σ is the input alphabet, (ii) Q is the (finite) set of states, (iii) $I \subseteq Q$ is the set of initial states, (iv) $T \subseteq Q \times Q$ is a *complete* transition relation, and (v) $L : Q \rightarrow 2^{\Sigma}$ is the labeling function that assigns to each state the set of atoms in Σ that are true in that state. We denote with $|M|$ the number of states in M, *i.e.*, $|Q|$. Given a path $\pi \coloneqq \langle q_0, q_1, \ldots \rangle$ in M, we denote with $L(\pi)$ the sequence $\langle L(q_0), L(q_1), \ldots \rangle$. The path π is called *initialized* iff $q_0 \in I$. The model checking problem takes as input a Kripke structure and a temporal formula, and asks to find whether all the initialized traces of the former satisfy the latter.

Definition 4 (The model checking problem). *Given a Kripke structure M and a linear temporal formula ϕ, the model checking problem is the problem of finding whether* all *the initialized traces π of M are such that $L(\pi) \models \phi$, written $M \models \mathsf{A}\phi$ (where A is the* "for all paths" *operator of* CTL*).*

Realizability and reactive synthesis are in some sense more ambitious problems than model checking, since they aim to find whether a given temporal formula ϕ over two sets $\mathcal{U}$ and $\mathcal{C}$ of uncontrollable and controllable variables, respectively, is implementable and, if this is the case, to synthesize a possible *implementation*. Usually, realizability is modeled as a two-player game between Environment, who tries to violate the specification and Controller, who tries to fulfill it. In this setting, an implementation of the specification is represented by a *strategy*.

Definition 5 (Strategies and Languages of Strategies). *Let $\mathcal{U}$ and $\mathcal{C}$ be two disjoint sets of* input *(or* uncontrollable*) and* output *(or* controllable*) variables, respectively. A* strategy *g is a function $g : (2^{\mathcal{U}})^+ \rightarrow 2^{\mathcal{C}}$. We define the* language *of the strategy g, denoted as $\mathcal{L}(g)$, as the set of all and only the sequences $\langle(\mathsf{U}_0 \cup \mathsf{C}_0), (\mathsf{U}_1 \cup \mathsf{C}_1), \ldots\rangle$ such that $\mathsf{U}_i \in 2^{\mathcal{U}}$ and $\mathsf{C}_i = g(\langle \mathsf{U}_0, \ldots, \mathsf{U}_i\rangle)$, for all $i \geq 0$.*

Definition 6 (Realizability and Synthesis for LTL). *Let ϕ be a temporal formula over the alphabet $\Sigma = \mathcal{U} \cup \mathcal{C}$, where $\mathcal{U}$ is the set of input variables, $\mathcal{C}$ the set of output variables, and $\mathcal{U} \cap \mathcal{C} = \varnothing$. We say that ϕ is* realizable *if and only if there exists a strategy $g : (2^{\mathcal{U}})^+ \rightarrow 2^{\mathcal{C}}$ such that $\mathcal{L}(g) \subseteq \mathcal{L}(\phi)$. If ϕ is realizable, the synthesis problem is the problem of computing such a strategy.*

The strategies which we are mainly interested in are the ones that can be represented finitely. In the literature, there are two main (and equivalent) representations for finite strategies, that is, *Mealy machines* and *Moore machines*. In this paper, we are mainly interested in the first ones.

Definition 7 (Mealy Machine). *A* Mealy machine *is a tuple $M = (\Sigma_{\mathcal{U}}, \Sigma_{\mathcal{C}}, Q, q_0, \delta)$ such that: (i) $\Sigma_{\mathcal{U}}$ and $\Sigma_{\mathcal{C}}$ are the input and output alphabets, respectively; (ii) Q is the (finite) set of states and q_0 is the initial state; (iii) $\delta : Q \times \Sigma_{\mathcal{U}} \rightarrow \Sigma_{\mathcal{C}} \times Q$ is the* total *transition* function*. We say that an infinite word $\sigma = \langle \sigma_0, \sigma_1, \ldots\rangle \in (\Sigma_{\mathcal{U}} \cup \Sigma_{\mathcal{C}})^{\omega}$ is* accepted *by M iff there exists a* trace *$\langle (q_0, \sigma_0), (q_1, \sigma_1), \ldots\rangle \in (Q \times (\Sigma_{\mathcal{U}} \cup \Sigma_{\mathcal{C}}))^{\omega}$ such that $\delta(q_i, \sigma_i \cap \Sigma_{\mathcal{U}}) = (\sigma_i \cap \Sigma_{\mathcal{C}}, q^{i+1})$, for all $i \geq 0$. We define the language of M, written as $\mathcal{L}(M)$, as the set of all the infinite words accepted by M.*

A fundamental feature is the *small model property* for realizability of LTL+P [11, 17, 22], which ensures that each realizable LTL+P formula has at least a finitely representable strategy.

Proposition 1 (Small model property of LTL+P [22]). *Let ϕ be an LTL+P formula and $n = |\phi|$. If ϕ is realizable by a strategy g, then there exists a Mealy machine M_g such that (i) M_g has at most $2^{2^{c \cdot n}}$ states, for some constant $c \in \mathbb{N}$, and (ii) $\mathcal{L}(M_g) \subseteq \mathcal{L}(\phi)$.*

3 $\mathsf{LTL}_{\mathsf{EBR}}$+P with fairness, assumptions, and guarantees

In this section, we extend $\mathsf{LTL}_{\mathsf{EBR}}$+P (see Def. 1) with fairness conditions (*i.e.*, of type $\mathsf{GF}\alpha$), assumptions and guarantees (that correspond to the antecedent

and the consequent of a logical implication). The syntax of the resulting logic, called GR-EBR, is the following.

Definition 8 (The logic GR-EBR). *The GR-EBR logic comprises all and only those formulas that can be written in the following form:*

$$(\psi^1_{ebr} \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\psi^2_{ebr} \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

where $m, n \in \mathbb{N}$, $\psi^1_{ebr}, \psi^2_{ebr} \in \mathsf{LTL_{EBR}{+}P}$ *and* $\alpha_i, \beta_j \in \mathsf{LTL_P}$, *for each* $i, j \in \mathbb{N}$.

3.1 Expressiveness of GR-EBR

Each $\mathsf{LTL_{EBR}{+}P}$ formula ϕ is a GR-EBR formula as well. In fact, $\phi \equiv (\top \wedge \top) \rightarrow (\phi \wedge \top) \in$ GR-EBR. It follows that any safety language definable in LTL+P is definable in GR-EBR as well. In addition, GR-EBR is *strictly* more expressive than $\mathsf{LTL_{EBR}{+}P}$, since the former can express also non-safety properties, like $\mathsf{G}(p) \rightarrow \mathsf{G}(q)$.

Consider the temporal hierachy defined by Manna and Pnueli in [19]. The Reactivity class is defined as the set of all and only those languages definable by formulas of type $\bigwedge_i(\mathsf{GF}\alpha_i \rightarrow \mathsf{GF}\beta_i)$ where each α_i and each β_i are pure-past LTL+P formulas. It is known that LTL+P is expressively equivalent to the Reactivity class. Moreover, if we fix the number of conjuncts of the formula above to be N, that the resulting class (called Reactivity(N)) strictly contains Reactivity(N-1) and is strictly contained in Reactivity(N+1). Compared to this classification, we have that GR-EBR is at least as expressive as the Reactivity(1) class, since each formula of type $\mathsf{GF}(\alpha) \rightarrow \mathsf{GF}(\beta)$ belongs to GR-EBR. However, the exact expressiveness of GR-EBR is still unknown.

On a more practical side, we found that some benchmarks of SYNTCOMP [15], like *simple_arbiter_N* (for each $N \in \{2, 4, 6, 8, 10, 12\}$) and also *escalator_bidirectional*, can be translated in GR-EBR with minor rewritings.

3.2 An Example

We take the example proposed in [4] and we extend it with fairness, assumptions and guarantees. Suppose that we want to synthesize an arbiter that, given a request from client i (for some $i \in \{1, \ldots, n\}$) in the environment, assigns the grant to the corresponding client, in such a way to guarantee the following properties: (1) *bounded response*: the grant is assigned at most k time units, for some $k > n$, after the request is issued; (2) *mutual exclusion*: the arbiter can assign a grant at most to one client at a time. The conjunction of the previous two requirements form the guarantees for the controller. The assumptions for the environment are the following: (1) initially, there are no requests; (2) if a request is issued at time i, then it cannot be issued until time $i + k$; (3) there are infinitely many requests from each client.

In order to write a specification of the arbiter, we can model the requests for the n clients with the (uncontrollable) variables $r_1, \ldots, r_n$. Similarly, the grant corresponding to the request r_i can be modeled with the (controllable) variable g_i, for each $i \in \{1, \ldots, n\}$. The assumption for the environment corresponds to the $\mathsf{LTL}_{\mathsf{EBR}}$+P formula ϕ_e defined as follows:

$$\bigwedge_{i=1}^{n} \neg r_i \ \wedge \ \bigwedge_{i=1}^{n} \mathsf{G}(r_i \rightarrow \mathsf{G}^{[1,k]} \neg r_i) \ \wedge \ \bigwedge_{i=1}^{n} \mathsf{GF} r_i$$

The guarantees for the controller correspond to the $\mathsf{LTL}_{\mathsf{EBR}}$+P formula ϕ_c defined as follows:

$$\bigwedge_{i=1}^{n} \mathsf{G}(r_i \rightarrow \mathsf{F}^{[0,k]} g_i) \ \wedge \ \mathsf{G}(\bigwedge_{1 \leq i < j \leq n} \neg (g_i \wedge g_j))$$

The overall specification is $\phi_e \rightarrow \phi_c$ and syntactically belongs to GR-EBR.

Our goal is to solve the realizability problem for GR-EBR specifications by reducing it to realizability subproblems for safety specifications. The reduction to safety, which we will give in Sec. 5, generates a safety formula for each integer k, in such a way to guarantee the following important properties: (i) *soundness*, ensuring that the realizability of the k^{th} subproblem implies the realizability of the starting formula, and (ii) *completeness*, establishing the existence of an upper bound μ such that the unrealizability of all the k^{th} subproblems with $k \leq \mu$ implies the unrealizability of the starting formula. In the next section we will give a general framework for (sound and) complete reductions. From it, in Sec. 5, we will derive one for GR-EBR specifications, showing also how the realizability of each safety subproblems can be solved symbolically.

4 A Framework of Safety Reductions for LTL+P Realizability

The central question of this section is: *how can we obtain a complete safety reduction for the realizability problem of specifications written in (fragments of) LTL?* In the following, we propose a framework to answer it.

The core and the main novelty of our framework is a link with safety reductions for model checking: in order to design a complete reduction for the realizability problem, one can prove that it is complete for the model checking problem and then use our framework to derive completeness for realizability. On one hand, this allows to prove completeness at the level of model checking, which is simpler than proving completeness for realizability. On the other hand, this opens the possibility of using existing safety reductions already devised for model checking for realizability as well. We start by defining what is a safety reduction in the context of our framework.

Definition 9 (Safety reduction). *Let $\mathcal{S} \subseteq$ LTL be a fragment of LTL. A* safety reduction for $\mathcal{S}$ *is a function $[\![\cdot]\!]$ such that, for each formula $\phi \in \mathcal{S}$ over*

the alphabet Σ, it holds that $[\![\phi]\!] = \{\phi_k\}_{k\in\mathbb{N}}$, where ϕ_k is a safety *formula over the alphabet Σ such that $\phi_k \rightarrow \phi$, for any $k \in \mathbb{N}$. With $[\![\phi]\!]^k$, we will denote the formula ϕ_k of the set above.*

Link between realizability and model checking The rationale behind the link between realizability and model checking is the following one: since we can easily view Mealy machines as (a particular type of) Kripke structures and viceversa, and since by Prop. 1 we can restrict realizability to the search of finite strategies representable by Mealy machines, the realizability problem of the $\mathsf{LTL{+}P}$ formula ϕ can be reduced to checking if there exists a Mealy machine M_g such that $M'_g \models \mathsf{A}\,\phi$, where M'_g is the Kripke structure *corresponding to* M_g.

The Kripke structure M'_g *corresponding to* the Mealy machine $M_g = (2^{\mathcal{U}}, 2^{\mathcal{C}}, Q, q_0, \delta)$ is defined as $M'_g = (2^{\mathcal{U}\cup\mathcal{C}}, Q', I', T', L')$ where:

1. $Q' = Q \times \{q_U \mid U \in 2^{\mathcal{U}}\} \times \{q_C \mid C \in 2^{\mathcal{C}}\}$;
2. $I' = \{(q_0, q_U, q_C) \in Q' \mid \delta(q_0, U) = (C, q') \text{ for any } U \in 2^{\mathcal{U}}, C \in 2^{\mathcal{C}} \text{ and } q' \in Q\}$,
3. $T' = \{((q, q_U, q_C), (q', q_{U'}, q_{C'})) \mid \delta(q, U) = (C, q') \text{ for any } U, U' \in 2^{\mathcal{U}}, C, C' \in 2^{\mathcal{C}}, \text{ and } q, q' \in Q'\}$ and
4. $L'((q, q_U, q_C)) = U \cup C$.

The Kripke structure M'_g is such that each trace of M'_g corresponds to a word of M_g, and viceversa.

In proving the completeness theorem, we will abstract from the concrete safety reduction and give the conditions for a general safety reduction $[\![\cdot]\!]$ (as defined in Def. 9) to be complete. These conditions are formalized in Def. 10.

Definition 10 (Sound and Complete safety reduction). *Let $\mathcal{S} \subseteq \mathsf{LTL}$ be a fragment of LTL, ϕ a formula in S, and $[\![\cdot]\!]$ a* safety reduction for $\mathcal{S}$. *We say that $[\![\cdot]\!]$ is* μ-complete*, for a given function $\mu : \mathbb{N} \rightarrow \mathbb{N}$ if and only if, for all $\phi \in \mathcal{S}$ and for all Kripke structures M:*

$$M \models \mathsf{A}\,\phi \quad \Leftrightarrow \quad \exists k \leq \mu(|M|)\,.\, M \models \mathsf{A}[\![\phi]\!]^k$$

We can finally state the main theorem of our framework, which uses Def. 10 and Prop. 1 in order to establish that if a safety reduction is complete for the model checking problem, then it is complete for the realizability problem as well.

Theorem 1 (Soundness and Completeness for $\mathsf{LTL{+}P}$ Realizability). *Let $\mathcal{S} \subseteq \mathsf{LTL}$ be a fragment of LTL, $\phi \in \mathcal{S}$ a formula over the input alphabet $\mathcal{U}$ and output alphabet $\mathcal{C}$ (with $n = |\phi|$) and $[\![\cdot]\!]$ a μ-complete safety reduction for $\mathcal{S}$, for a given function μ. It holds that:*

$$\phi \text{ is realizable} \quad \Leftrightarrow \quad \exists k \leq \mu(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c\cdot n}})\,.\, [\![\phi]\!]^k \text{ is realizable}$$

Proof. We first prove the *soundness*, which corresponds to the right-to-left direction. Suppose there exist a $k \leq \mu(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c\cdot n}})$ such that $[\![\phi]\!]^k$ is realizable.

Then, there exists a strategy $g : (2^{\mathcal{U}})^+ \rightarrow 2^{\mathcal{C}}$ such that $\mathcal{L}(g) \subseteq \mathcal{L}([\![\phi]\!]^k)$. By Prop. 1, there exists a Mealy machine $M_g = (2^{\mathcal{U}}, 2^{\mathcal{C}}, Q, q_0, \delta)$ with input alphabet $2^{\mathcal{U}}$ and output alphabet $2^{\mathcal{C}}$ such that $\mathcal{L}(M_g) \subseteq \mathcal{L}([\![\phi]\!]^k)$. Starting from M_g, let $M'_g = (2^{\mathcal{U}\cup\mathcal{C}}, Q', I', T', L')$ be the *corresponding* Kripke structure. The Kripke structure M'_g is such that each trace of M'_g corresponds to a word of M_g, and viceversa. Therefore all the traces π of M'_g are such that $L'(\pi) \models [\![\phi]\!]^k$, that is $M'_g \models \mathsf{A}[\![\phi]\!]^k$. Since by hypothesis $[\![\cdot]\!]$ is a μ-complete safety reduction, by Def. 10, it holds that $M'_g \models \mathsf{A}\,\phi$. This means that also $\mathcal{L}(M_g) \subseteq \mathcal{L}(\phi)$. Since M_g is a Mealy machine, this implies that ϕ is realizable.

We now prove *completeness*, which corresponds to the left-to-right direction. Suppose that ϕ is realizable. Since $\phi \in S$ and since $S \subseteq \mathsf{LTL}$, ϕ is an LTL formula as well. Therefore, by Prop. 1, there exists a Mealy machine M_g with input alphabet $2^{\mathcal{U}}$ and output alphabet $2^{\mathcal{C}}$ such that $\mathcal{L}(M_g) \subseteq \mathcal{L}(\phi)$ with at most $2^{2^{c \cdot n}}$ states, for some constant $c \in \mathbb{N}$. From M_g, we build an equivalent Kripke structure M'_g with input alphabet $\Sigma' = 2^{\mathcal{U}\cup\mathcal{C}}$, as described above for the soundness proof. It holds that $M'_g \models \mathsf{A}\,\phi$. Since by hypothesis $[\![\cdot]\!]$ is a μ-complete safety reduction for S, and since $|Q'| = 2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot |Q|$ (where Q and Q' are the set of states of M_g and M'_g, respectively), by Def. 10, there exists a $k \leq \mu(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c \cdot n}})$ such that $M'_g \models \mathsf{A}[\![\phi]\!]^k$. This means that also $\mathcal{L}(M_g) \subseteq \mathcal{L}([\![\phi]\!]^k)$. Since M_g is a Mealy machine, this means that there exists a $k \leq \mu(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c \cdot n}})$ such that $[\![\phi]\!]^k$ is realizable. □

Novelty and Usage As already mentioned before, a distinguished and important feature of our framework is that it provides a link with safety reductions for the *model checking problem*. This opens the possibility to use model checking safety reductions for the realizability problem as well, provided that the reduction fulfills the requirements in Def. 10. In the next sections, we will define a *concrete* safety reduction for GR-EBR specifications that is *complete* with respect to Def. 10, and we will use it for introducing a novel algorithm for GR-EBR realizability. Using Theorem 1, we will derive a corollary for the completeness of our algorithm.

In practice The upper bound for the value of $\mu(\cdot)$ (after which we can answer *unrealizable*) is doubly exponential in the size of the initial formula and therefore, in practice, it is prohibitively large. It follows that usually the *completeness* of a safety reduction can be exploited in practice only for making sure that, starting from a *realizable* specification, we will eventually find a $k \in \mathbb{N}$ such that the k^{th} subproblem is realizable. Therefore, like K-Liveness for model checking [6], we can use our algorithm in parallel with another one that checks for the unrealizability of the specification. The first that terminates stops the other and, thus, the entire procedure. We remark that we cannot check the unrealizability of ϕ by solving the dualized game (*i.e.*, looking for a Moore-type strategy of Environment) for $\neg\phi$, because GR-EBR and $\mathsf{LTL}_{\mathsf{EBR}}$+P are *not* closed under complementation.

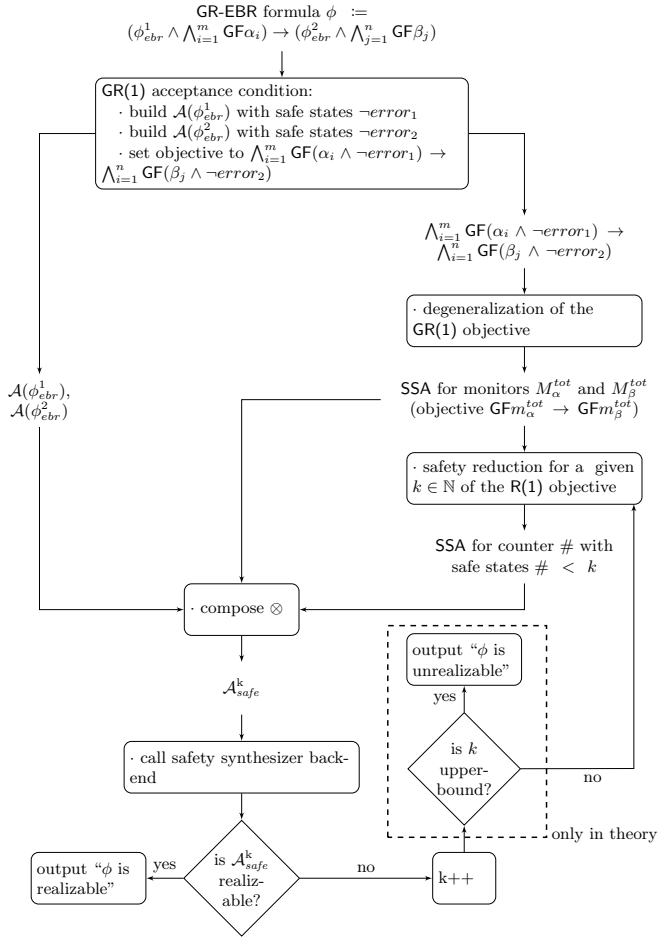

Fig. 1: Low-level view of the procedure for the realizability of GR-EBR formulas.

5 A Safety Reduction for GR-EBR

In this section, we describe the algorithm for solving realizability of GR-EBR specifications. It consists in three steps. Firstly, we build the product between the two symbolic and safety automata for the safety parts of both assumptions and guarantees. This product automaton has a GR(1) accepting condition. The second step consists in a so-called *degeneralization*, that, by using deterministic monitors, turns the GR(1) accepting condition into a Reactivity(1) (R(1), for short) condition. The third and last step, that is the core of the procedure, reduces the realizability problem over the above automaton to a *sequence* of safety synthesis problems, that is, realizability problems over safety (and symbolic) automata $\mathcal{A}^{\mathrm{k}}_{safe}$, one for each index $k \in \mathbb{N}$. By introducing a *concrete* safety reduction $[\![\cdot]\!]_{ebr}$ for GR-EBR, and by proving that it is *complete* with respect to

Def. 10, we prove the completeness of the entire procedure. The structure of the full procedure is depicted in Fig. 1.

Finally, note that, as for now, there is no incrementality between an iteration and the next one, because of the lack of incremental safety synthesizers. The only point that we save between one iteration and the next one is the construction of the two symbolic safety automata, which is performed only once during the procedure.

5.1 Construction of the automaton with a GR(1) condition

In this part, we describe the first step of the algorithm. Starting from a GR-EBR formula $\phi := (\phi^1_{ebr} \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\phi^2_{ebr} \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$, the objective is to obtain an automaton $\mathcal{A}$ such that: (i) it has a GR(1) accepting condition, and (ii) it recognizes the same language of ϕ, *i.e.*, $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A})$. In order to do that, we first build the two symbolic safety automata for the safety parts of both the assumptions and the guarantees, that is for ϕ^1_{ebr} and ϕ^2_{ebr}. Since by definition both are $\mathsf{LTL}_{\mathsf{EBR}}$+P formulas, we use the transformation described in [4], to which the reader is referred for more details.

From now on, let $\mathcal{A}(\phi^1_{ebr})$ and $\mathcal{A}(\phi^2_{ebr})$ be the automata for ϕ^1_{ebr} and ϕ^2_{ebr}, respecively. Let $\mathcal{A}_{\phi_{\text{ebr}}}$ be the product automaton $\mathcal{A}(\phi^1_{ebr}) \times \mathcal{A}(\phi^2_{ebr})$. The question is how to set the acceptance condition of $\mathcal{A}_{\phi_{\text{ebr}}}$ such that the conditions (i) and (ii) of above are fulfilled. We answer this question by examining how the automata $\mathcal{A}(\phi^1_{ebr})$ and $\mathcal{A}(\phi^2_{ebr})$ are made internally. Take for example the formula $\mathsf{G}p$ (for some atomic proposition $p \in \Sigma$). The safety automaton corresponding to this formula comprises an *error bit* as one of its state variables, let us call it `error`, which is initially set to be `false`. The transition function for `error` is deterministic and updates `error` to `true` if $\neg p$ holds in the current state, or keeps its value otherwise. The set of safe states comprises all and only those states in which `error` is `false`. In a symbolic setting, this is expressed by the formula $\mathsf{G}\neg$`error`. In this way, p is forced to hold constantly in all (and only) the words accepted by the automaton.

A crucial property of each error bit is *monotonicity*: once `error` is set to `true`, it can never be set to `false` again. Formally, given a trace τ of the automaton, it holds that, if there exists $i \geq 0$ such that $\tau(i) \models$ `error`, then $\tau(j) \models$ `error`, for all $j \geq i$. Monotonicity of the error bits allows us to express an accepting condition of type $\mathsf{G}\neg$`error` in terms of $\mathsf{GF}\neg$`error`, by maintaining the equivalence.

Lemma 1 (Monotonicity of Error Bits). *Each error bit is monotone.*

Proof. Consider a trace τ of an automaton with an accepting condition of the type $\mathsf{G}\neg$`error`. If $\tau \models \mathsf{G}\neg$`error` then of course $\tau \models \mathsf{GF}\neg$`error`. Suppose now that $\tau \models \mathsf{GF}\neg$`error`. If by contradiction we suppose that $\tau \not\models \mathsf{G}\neg$`error`, we have that there exists an $i \geq 0$ such that $\tau(i) \models$ `error`. By the monotonicity property, this would mean that also $\tau(j) \models$ `error`, for all $j \geq i$, that is $\tau \models \mathsf{FG}$`error`, but this a contradiction with out hypothesis. Therefore, we proved that changing the acceptance condition of an automaton from a $\mathsf{G}\neg$`error` to $\mathsf{GF}\neg$`error` maintains the equivalence. □

Let $\texttt{error}_1$ and $\texttt{error}_2$ be the error bits of $\mathcal{A}(\phi^1_{ebr})$ and $\mathcal{A}(\phi^2_{ebr})$, respectively. Let $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$ be the automaton obtained from $\mathcal{A}_{\phi_{\text{ebr}}}$ by replacing its acceptance condition with the following $\mathsf{GR(1)}$ condition:

$$(\mathsf{GF}\neg\texttt{error}_1 \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\mathsf{GF}\neg\texttt{error}_2 \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j) \tag{1}$$

The intuition is that $\texttt{error}_1$ and $\texttt{error}_2$ keep track of the *safety* parts of ϕ, that is ϕ^1_{ebr} and ϕ^2_{ebr}. The following lemma proves the equivalence between ϕ and $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$.

Lemma 2. *Let ϕ be an GR-EBR formula. It holds that $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}})$.*

Proof. Let $\phi \in \mathsf{GR\text{-}EBR}$. ϕ is of the following form:

$$(\phi^1_{ebr} \rightarrow \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\phi^2_{ebr} \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$$

By the theorems proved in [4], it holds that $\mathcal{L}(\phi^1_{ebr}) = \mathcal{L}(\mathcal{A}(\phi^1_{ebr}))$ and $\mathcal{L}(\phi^2_{ebr}) = \mathcal{L}(\mathcal{A}(\phi^2_{ebr}))$.

Consider first the left-to-right direction. Let $\sigma \in \mathcal{L}(\phi)$. We prove that $\sigma \in \mathcal{L}(\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}})$. Each $\sigma \in \mathcal{L}(\phi)$ is such that: a. either $\sigma \models \neg\phi^1_{ebr} \vee \neg(\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i)$, b. or $\sigma \models \phi^2_{ebr} \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$ Recall that $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$ is defines as the product automaton $\mathcal{A}(\phi^1_{ebr}) \times \mathcal{A}(\phi^2_{ebr})$ with the acceptance condition α defined as $(\mathsf{GF}\neg\texttt{error}_1 \wedge \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i) \rightarrow (\mathsf{GF}\neg\texttt{error}_1 \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j)$.

Consider case *a.* If $\sigma \models \neg\phi^1_{ebr} \vee \neg(\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i)$, then *the* trace induced by σ in $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$ is such that at least one of the following two cases hold:

a.1. either $\exists i \geq 0$ such that $\tau(i) \models \texttt{error}_1$, that is $\tau \models \mathsf{F}(\texttt{error}_1)$. In this case, we exploit *monotonicity* of $\texttt{error}_1$. Since $\tau \models \mathsf{F}(\texttt{error}_1)$, it also holds that $\tau \models \mathsf{FG}(\texttt{error}_1)$, that is $\tau \not\models \mathsf{GF}(\neg\texttt{error}_1)$. As a consequence, $\tau \models \alpha$, where α is the acceptance condition of $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$, and thus $\sigma \in \mathcal{L}(\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}})$.

a.2. or $\tau \models \neg\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i$. In this case, of course, $\tau \models \alpha$ (that is, τ satisfies the acceptance condition of $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$), and thus $\sigma \in \mathcal{L}(\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}})$.

Consider now the case *b.* If $\sigma \models \phi^2_{ebr} \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$, then $\sigma \models \phi^2_{ebr}$ and $\sigma \models \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$. Therefore, *the* trace induced by σ in $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$ is such that $\tau \models \mathsf{G}(\neg\texttt{error}_2) \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$, that implies that $\tau \models \mathsf{GF}(\neg\texttt{error}_2) \wedge \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$. Therefore, $\tau \models \alpha$, and thus $\sigma \in \mathcal{L}(\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}})$. The opposite direction can be proved similarly. □

5.2 Degeneralization

The objective of this part is to transform the $\mathsf{GR(1)}$ accepting condition of the automaton $\mathcal{A}^{\wedge\rightarrow\wedge}_{\text{ebr}}$, that is of the form $\bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$, into a condition of the form $\mathsf{GF}\alpha \rightarrow \mathsf{GF}\beta$ (also called *Reactivity(1)* objective, $\mathsf{R(1)}$, for short). In this context, we will use the term *monitor* as a synonym of *deterministic*

automaton. In order to accomplish the task, for each α_i (resp. for each β_i), we define a monitor M_{α_i} (resp. M_{β_i}) that is set to *true* when α_i (resp. β_i) has been read and is reset to *false* when all the α_i (resp. β_i) have been read. For this last condition, we define the monitors M_α^{tot} and M_β^{tot}.

Let M_{α_i} and M_α^{tot} be the symbolic safety automata such that their input alphabet is 2^Σ (where Σ is the alphabet of the starting GR-EBR formula), their set of *state variables* are $\{m_{\alpha_i}\}$ and $\{m_\alpha^{tot}\}$, respectively, all their reachable states are safe states, and their transition relations are the following:

$\mathrm{init}(m_{\alpha_i}) := 0$	$\mathrm{init}(m_\alpha^{tot}) := 0$
$\mathrm{next}(m_{\alpha_i}) :=$ case	$\mathrm{next}(m_\alpha^{tot}) :=$ case
$\alpha_i : 1$	$m_{\alpha_1} \wedge \cdots \wedge m_{\alpha_m} : 1$
$m_\alpha^{tot} : 0$	default : 0
default : m_{α_i}	esac
esac	

We define M_{β_i} and M_β^{tot} as M_{α_i} and M_α^{tot}, respectively, but with α_i substituted with β_i and α substituted with β. Let $\mathcal{A}_{degen}$ be the product between all the M_{α_i}, M_{β_i}, M_α^{tot} and M_β^{tot}. Let $\mathcal{A}_{\text{degen}}^{\mathsf{GF}\to\mathsf{GF}}$ be the automaton obtained from $\mathcal{A}_{degen}$ by replacing its accepting condition with the Reactivity(1) condition $\mathsf{GF}m_\alpha^{tot} \to \mathsf{GF}m_\beta^{tot}$. We can prove the following lemma, which states that this step of the algorithm maintains the equivalence.

Lemma 3. $\mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge\to\wedge}) = \mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{\text{degen}}^{\mathsf{GF}\to\mathsf{GF}})$.

Proof. We prove separately the two directions. Consider first the right-to-left direction. Let σ be an infinite word of $\mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{\text{degen}}^{\mathsf{GF}\to\mathsf{GF}})$. Then σ is a word in $\mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}})$. Moreover, σ is a word in $\mathcal{L}(\mathcal{A}_{\text{degen}}^{\mathsf{GF}\to\mathsf{GF}})$ and thus there exists a run τ induced by σ such that $\tau \models \mathsf{GF}m_\alpha^{tot} \to \mathsf{GF}m_\beta^{tot}$, that is, $\tau \models \mathsf{FG}\neg m_\alpha^{tot} \vee \mathsf{GF}m_\beta^{tot}$. We divide in cases:

- if $\tau \models \mathsf{FG}\neg m_\alpha^{tot}$, then by the semantics of the temporal operators F and G, there exists an $i \geq 0$ such that for all $j \geq i$, $\tau_j \models \neg m_\alpha^{tot}$. By construction of the monitors m_α^{tot}, this means that there exists an $i \geq 0$ such that for all $j \geq i$, $\tau_j \models \bigvee_{k=1}^m \neg m_{\alpha_k}$. This implies that, there exists a $k \in [1, m]$ and an $i \geq 0$ such that for all $j \geq i$, such that $\tau_j \models \neg m_{\alpha_k}$. Indeed, suppose by contradiction that it is not so: then for all $k \in [1, m]$, there exists infinitely many positions $i \geq 0$ such that $\tau_i \models m_{\alpha_k}$. This would mean that the monitor M_α^{tot} is set to *true* infinitely many times, that is $\mathsf{GF}m_\alpha^{tot}$, but this is a contradiction with our hypothesis. Therefore, it holds that $\tau \models \bigvee_{k=1}^m \mathsf{FG}\neg m_{\alpha_k}$, and thus also that $\tau \models \bigwedge_{i=1}^m \mathsf{GF}\alpha_i \to \bigwedge_{j=1}^n \mathsf{GF}\beta_j$. Overall, since τ is induced by σ, we have that σ is a word of $\mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}})$ that induces a run τ such that $\tau \models \bigwedge_{i=1}^m \mathsf{GF}\alpha_i \to \bigwedge_{j=1}^n \mathsf{GF}\beta_j$, that is $\sigma \in \mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge\to\wedge})$.
- If otherwise $\tau \models \mathsf{GF}m_\beta^{tot}$, then there exists infinitely many positions $i \geq 0$ such that $\tau_i \models m_\beta^{tot}$. Moreover, it holds that for all $i_1 \geq 0$ and for all $i_2 \geq i_1$, if $\tau_{i_1} \models m_\beta^{tot}$ and $\tau_{i_2} \models m_\beta^{tot}$, then, for all $1 \leq k \leq n$, there exists a $i_1 \leq j \leq i_2$ such that $\tau_j \models m_{b_k}$. Putting together these two points, we have that for all $1 \leq k \leq n$, there exists infinitely many $i \geq 0$ such that $\tau_i \models m_{b_k}$. That is, $\tau \models \bigwedge_{k=1}^n \mathsf{GF}m_{b_k}$. By definition of the monitors M_{β_i} and since τ is induced

by σ, we have that σ is a word in $\mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}})$ that induces a run τ such that $\tau \models \bigwedge_{i=1}^{m} \mathsf{GF}\alpha_i \rightarrow \bigwedge_{j=1}^{n} \mathsf{GF}\beta_j$. That is, $\sigma \in \mathcal{L}(\mathcal{A}_{\text{ebr}}^{\wedge\rightarrow\wedge})$.

The proof the left-to-right direction is specular, and therefore is omitted from the presentation. □

5.3 Reduction to Safety for Reactivity(1) objectives

In this part, we describe a *complete* safety reduction (see Def. 10) tailored for Reactivity(1) objectives. We will apply this reduction on the automaton $\mathcal{A}_{\text{degen}}^{\mathsf{GF}\rightarrow\mathsf{GF}}$ obtained from the previous step. The intuition is to use a *counter* to count and limit the number of positions, after a position in which m_β^{tot} holds, in which $m_\alpha^{tot} \wedge \neg m_\beta^{tot}$ holds. We define the counter as follows.

Definition 11 (Counter for the Reactivity(1) objective). *Let $\mathcal{A}^k_{\#^{\rightarrow}_{\alpha,\beta}}$ be the symbolic and deterministic safety automaton whose set of safe states is represented by the formula $\mathsf{G}(\#^{\rightarrow}_{\alpha,\beta} < k)$ and whose transition relation is the following:*

$init(\#^{\rightarrow}_{\alpha,\beta}) := 0$
$next(\#^{\rightarrow}_{\alpha,\beta}) := case$
$\quad m_\beta^{tot} : 0$
$\quad m_\alpha^{tot} : \#^{\rightarrow}_{\alpha,\beta} + 1$
$\quad default : \#^{\rightarrow}_{\alpha,\beta}$
$esac$

We define $\mathcal{A}^{\text{k}}_{safe} \coloneqq \mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{degen} \times \mathcal{A}^k_{\#^{\rightarrow}_{\alpha,\beta}}$, and we set the accepting condition of $\mathcal{A}^{\text{k}}_{safe}$ to be the one of $\mathcal{A}^k_{\#^{\rightarrow}_{\alpha,\beta}}$, *i.e.*, $\mathsf{G}(\#^{\rightarrow}_{\alpha,\beta} \leq \#^{\rightarrow}_{\alpha,\beta} < k)$. The automaton $\mathcal{A}^{\text{k}}_{safe}$ is a symbolic and deterministic safety automaton, and therefore it can be used as an arena for a safety game. In practice, we check the realizability of $\mathcal{A}^{\text{k}}_{safe}$ by means of a tool for safety synthesis. We start with $k = 0$, and we check the realizabilty of $\mathcal{A}^{\text{k}}_{safe}$: if Controller has a strategy, than we stop, otherwise we increment k and we repeat the cycle.

In order to prove that this step is sound and complete, we use the framework described in Sec. 4. We call $[\![\cdot]\!]_{ebr}$ the safety reduction described in this part. Since the framework works with formulas rather than with automata, for all $\phi \in$ GR-EBR, we define $[\![\phi]\!]^k_{ebr}$ to be any *safety formula* such that $\mathcal{L}([\![\phi]\!]^k_{ebr}) = \mathcal{L}(\mathcal{A}^{\text{k}}_{safe})$. From now, with $\mathsf{id} : \mathbb{N} \rightarrow \mathbb{N}$ we denote the *identity* function.

Theorem 2. *$[\![\cdot]\!]_{ebr}$ is a id-complete safety reduction for GR-EBR.*

Proof. We have to prove that, for all $\phi \in$ GR-EBR, for all Kripke structures M and for all $k \in \mathbb{N}$, it holds that:

$$M \models \mathsf{A}\,\phi \qquad \Leftrightarrow \qquad \exists k \leq \mathsf{id}(|M|)\ .\ M \models \mathsf{A}[\![\phi]\!]^k_{ebr}$$

We prove separately the two directions. Consider first the *soundness* which corresponds to the right-to-left direction. Suppose that $M \models \mathsf{A}[\![\phi]\!]^k_{ebr}$. It holds

that, for each initialized trace π of M, $L(\pi) \models [\![\phi]\!]^k_{ebr}$, where $L(\cdot)$ is the labeling function of M. Let π be an initialized trace of M. By definition of $[\![\cdot]\!]_{ebr}$, it holds that, there exists a run τ induced by $L(\pi)$ such that: (i) τ is accepting in $\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{degen}$, and (ii) τ is accepting in $\mathcal{A}^k_{\#^{\rightarrow}_{\alpha,\beta}}$. From the second point, we have that:

- either, $\#^{\rightarrow}_{\alpha,\beta}$ make infinitely many *resets*. This means that there exists infinitely many positions in τ in which m^{tot}_{α} holds and, after at most k occurrences of m^{tot}_{α}, there is a m^{tot}_{β}. Therefore, in particular, there exists infinitely many positions in which m^{tot}_{β} holds, that is $\tau \models \mathsf{GF} m^{tot}_{\beta}$.
- or the counter $\#^{\rightarrow}_{\alpha,\beta}$ stops to increment because, because it does not read any m^{tot}_{α}. This means that there exists *finitely* many positions in which m^{tot}_{α} holds, that is $\tau \models \mathsf{FG} \neg m^{tot}_{\alpha}$.

Therefore, it holds that $\tau \models \mathsf{FG} \neg m^{tot}_{\alpha} \vee \mathsf{GF} m^{tot}_{\beta}$, that is $\tau \models \mathsf{GF} m^{tot}_{\alpha} \rightarrow \mathsf{GF} m^{tot}_{\beta}$. Finally, we have that τ is an accepting run of $\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{degen}$ such that $\tau \models \mathsf{GF} m^{tot}_{\alpha} \rightarrow \mathsf{GF} m^{tot}_{\beta}$. Since by hypothesis $L(\pi)$ is induced by τ, by definition of $\mathcal{A}^{\mathsf{GF}\rightarrow\mathsf{GF}}_{\text{degen}}$, we have that $L(\pi) \in \mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}^{\mathsf{GF}\rightarrow\mathsf{GF}}_{\text{degen}})$. By concatenating Lemma 2 and Lemma 3, we have that $L(\pi) \in \mathcal{L}(\phi)$, and therefore $\pi \models \phi$. It follows that $M \models \mathsf{A}\,\phi$.

We now prove *completeness*, which corresponds to the left-to-right direction. Suppose that $M \models \mathsf{A}\,\phi$, where $\phi \in \mathsf{GR\text{-}EBR}$. We prove this case by contradiction. Suppose therefore that for all $k \leq \mathsf{id}(|M|)$, $M \not\models \mathsf{A}[\![\phi]\!]^k_{ebr}$. This means that there exists an initialized trace π in M such that $L(\pi) \notin \mathcal{L}([\![\phi]\!]^k_{ebr})$, for all $k \leq \mathsf{id}(|M|)$. By definition of $[\![\cdot]\!]_{ebr}$, for $k = \mathsf{id}(|M|)$, we have that *for all* runs τ induced by $L(\pi)$ in $\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}_{degen} \times \mathcal{A}^k_{\#^{\rightarrow}_{\alpha,\beta}}$, it holds that $\tau \not\models \mathsf{G}(\#^{\rightarrow}_{\alpha,\beta} \leq k)$. Let τ be one of these runs. There exists a position i in τ such that $\tau_i \models (\#^{\rightarrow}_{\alpha,\beta} = v)$, for some $v > k$. By definition of the counter $\#^{\rightarrow}_{\alpha,\beta}$, the run τ is such that:

$$\exists 0 < h_1 < h_2 < \cdots < h_v \,.\, (\quad \tau_{h_1} \models m^{tot}_{\alpha} \wedge \tau_{h_2} \models m^{tot}_{\alpha} \wedge \ldots \tau_{h_v} \models m^{tot}_{\alpha} \wedge$$
$$\forall h_1 \leq h \leq h_v \,.\, (\tau_j \models \neg m^{tot}_{\beta}))$$

Recall that τ is a run induced by $L(\pi)$. Since $v > k$, $k = \mathsf{id}(|M|)$ and M is a *finite-state* Kripke structure, the positions $h_1 \ldots h_v$ in π (attention: *not* in τ) cannot be all different. That is, there exists at least two indexes $s, e \in \mathbb{N}$ such that: (i) $1 \leq s < e \leq v$, (ii) $\pi_{h_s} = \pi_{h_e}$, and (iii) $\pi_{h_s} \models m^{tot}_{\alpha}$. Starting from π, we can build a *looping trace* π' that agrees with π in the prefix $\pi_{[0,h_e]}$ and then loops on the interval $\pi_{[h_s,h_e]}$. It holds that π' is an initialized trace of M and it induces a run τ' such that $\tau' \models \mathsf{GF} m^{tot}_{\alpha} \wedge \mathsf{FG} \neg m^{tot}_{\beta}$, that is $\tau' \not\models \mathsf{GF} m^{tot}_{\alpha} \rightarrow \mathsf{GF} m^{tot}_{\beta}$. Nevertheless, since $M \models \mathsf{A}\,\phi$, by Lemma 2 and Lemma 3, we have that $L(\pi') \in \mathcal{L}(\mathcal{A}_{\phi_{\text{ebr}}} \times \mathcal{A}^{\mathsf{GF}\rightarrow\mathsf{GF}}_{\text{degen}})$, and therefore this is a contradiction. This means that it has to hold that $L(\pi) \in \mathcal{L}([\![\phi]\!]^k_{ebr})$, that is $\pi \models [\![\phi]\!]^k_{ebr}$ for all the initialized traces π of M, and thus there exists a $k \leq \mathsf{id}(|M|)$ such that $M \models \mathsf{A}[\![\phi]\!]^k_{ebr}$. $\square$

With Theorem 1, we derive the following corollary that proves the completeness of our procedure.

Corollary 1. *For any formula $\phi \in$ GR-EBR, it holds that: ϕ is realizable iff $\exists k \leq \mathsf{id}(2^{|\mathcal{U}|} \cdot 2^{|\mathcal{C}|} \cdot 2^{2^{c \cdot n}})$ such that $[\![\phi]\!]^k_{ebr}$ is realizable.*

6 Experimental Evaluation

We implemented the algorithm described in Sec. 5 and summarized in Fig. 1 in a prototype tool called GRACE (which stands for *GR-ebr reAlizability ChEcker*) [4] . We chose SAFETYSYNTH [14] as a BDD-based backend for solving each safety game.

As competitor tools, we chose BoSy [9, 10, 12] and Strix [18, 20]. BoSy implements the Bounded Synthesis approach (see the paragraph on the related works in Sec. 1), while Strix is based on parity games and is the winner of SYNTCOMP 2018, 2019 and 2020. We set a timeout of 180 seconds. The experiments have been run on a 16-cores machine with a 2696.6 MHz AMD core with 62 GB of RAM.

We remark that a comparison with GR(1) synthesis tools is nontrivial. The majority of the tools for GR(1) only support the realizability of the *strict* implication (see for example [8]), not the standard one (which is our case). Therefore, although the latter can be reduced to the former [1], a non-trivial practical effort is required to write an algorithm for this translation.

We considered benchmarks of two types: (i) artificial, and (ii) derived from the SYNTCOMP [14] benchmarks'set. Regarding the artifical benchmarks, we partitioned them in four categories, each containing 30 benchmarks scalable in their dimension N, for a total of 120 formulas. The categories are the following ones:

1. $\mathsf{G}(u_0 \rightarrow \mathsf{X}(u_1 \rightarrow \mathsf{X}(u_2 \rightarrow \cdots \rightarrow \mathsf{X}(u_N)\dots))) \rightarrow \mathsf{G}(\bigwedge_{i=1}^{N}(u_i \leftrightarrow \mathsf{X}c_i))$
2. $(\mathsf{G}(u_0 \rightarrow \mathsf{X}(u_1 \rightarrow \mathsf{X}(u_2 \rightarrow \cdots \rightarrow \mathsf{X}(u_N)\dots))) \wedge \mathsf{X}^N\mathsf{G}u_N \wedge \mathsf{GF}u_N) \rightarrow (\bigwedge_{i=1}^{N}(u_i \leftrightarrow \mathsf{X}^N c_i) \wedge \mathsf{GF}c_N)$
3. $(\mathsf{G}(u_0) \wedge \mathsf{XG}(u_1) \wedge \cdots \wedge \mathsf{X}^N\mathsf{G}(u_N) \wedge \bigwedge_{i=1}^{N}\mathsf{GF}u_i) \rightarrow (\bigwedge_{i=1}^{N}\mathsf{G}(u_i \leftrightarrow c_i) \wedge \bigwedge_{i=1}^{N}\mathsf{GF}c_i)$
4. $(\neg u_0 \wedge \mathsf{G}^{[0,N]}\neg u_0 \wedge \mathsf{X}^{N+1}\mathsf{G}u_0) \rightarrow (\bigwedge_{i=1}^{N}\mathsf{G}(u_0 \leftrightarrow \mathsf{X}c_i) \wedge \bigwedge_{i=1}^{N}\mathsf{GF}(c_i \wedge u_0))$

The variables starting with u are $u$ncontrollable, while those starting with c are $c$ontrollables. All the benchmarks are realizable, and were specifically crafted to elicit potential criticalities of GRACE. In particular, the benchmarks in the fourth category have been specifically designed in order to force the minimum k of the termination of GRACE to increase with their dimension.

Regarding the benchmarks derived from the SYNTCOMP benchmarks'set, we included (i) *simple_arbiter_N* (for each $N \in \{2, 4, 6, 8, 10, 12\}$), *escalator_bidirectional*, which belong to the SYNTCOMP benchmarks' set, and (ii) our example for an arbiter (Sec. 3.2), with $N \in \{1, \dots, 15\}$.

Fig. 2 show the comparison between the tools on all the benchmarks of both types. All times are in seconds. From Fig. 2 (left), it is clear the exponential

[4] `https://es-static.fbk.eu/tools/grace/`

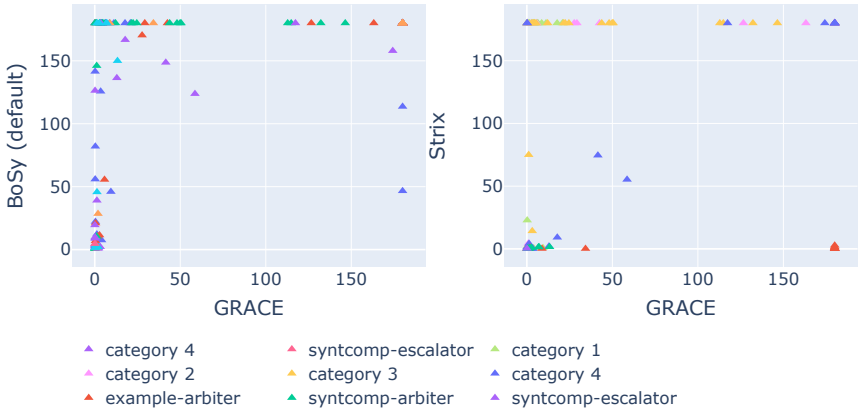

Fig. 2: GRACE compared to BoSy (on the left) and to Strix (on the right)

blowup in solving time in which BoSy occurs. The blowup involves formulas of both types, and of all four categories (of artificial benchmarks). For example, (i) on category 4, the solving times of BoSy on $N = 13, 14$ are 19.4 and 136.3 sec., respectively, and the corresponding automata have 27 and 31 states, respectively. (ii) on `simple_arbiter_N`, BoSy takes 45.714 sec. for $N = 8$, and reaches the timeout for $N = 10$. Fig. 3 (left), which compares the dimension of the benchmarks (X axis) with the solving time of GRACE, BoSy and Strix (Y axis), clearly shows this trend. A more precise study of the complexity of BoSy shows that the majority of the time spent by it is due to the construction of the UCW corresponding to the input formula, which is the task of the tools LTL3BA and SPOT. On the contrary, it is clear from Fig. 2 (left) that GRACE avoids this

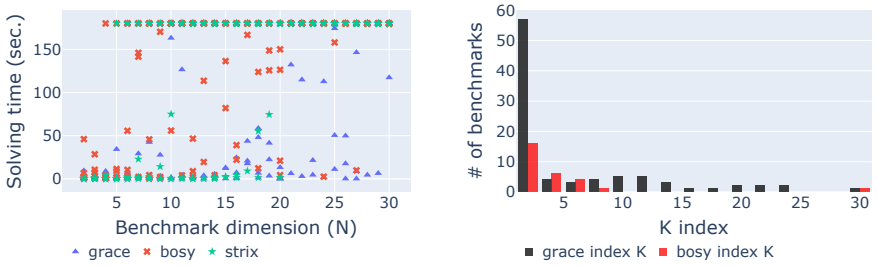

Fig. 3: On the left, the size of the benchmarks compared to solving time. On the right, GRACE vs BoSy on number of safety sub-problems.

bad behavior, most likely due to the fact that the explicit state UCW is never built. Similar considerations can be made for the tool Strix (see Fig. 2, right), except for the category `example-arbiter`, in which the solving times of Strix are consistently better than the ones of grace. A careful study revealed that all these benchmarks are transformed to the equi-realizable formula `true` by the preprocessor of Owl [16] (a tool for ω-automata manipulation), which Strix is based on.

The plot in Fig. 3 (right) shows, for each index k ranging from 1 to 31 (these correspond to the columns), on how many benchmarks (of both types) grace or BoSy terminate with index k (this corresponds to the height of a column). The benchmarks in category 4 and the ones of `simple_arbiter_N` force grace to terminate with increasing values of k. The plot in Fig. 3 points out that BoSy does not incur in this growth, except for one benchmark. Nevertheless, the solving times of grace are still better than the ones of BoSy. This witnesses the fact that each safety sub-problem generated by grace is very simple to solve.

7 Conclusions

In this paper, we introduced the logic of GR-EBR, an extension of $\mathsf{LTL}_{\mathsf{EBR}}$+P [4] adding fairness conditions and assumes-guarantees formulas, and studied its realizability problem. We aim at extending the work done in three directions: (i) as far as we know, there are no safety synthesizer (like safetysynth) that are able to exploit *incrementality*; since in our context, the only part of the automaton that changes between one iteration and the next one is the counter, some work may be saved; (ii) since GR(1) is a very efficient fragment, it is important to investigate whether there is a compilation from GR-EBR to GR(1); (iii) last but not least, we aim at giving a semantic characterization of GR-EBR, and at exploiting the proposed framework for more expressive logics, such as full LTL.

References

1. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Saar, Y.: Synthesis of reactive (1) designs. Journal of Computer and System Sciences **78**(3), 911–938 (2012)
2. Buchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. In: The Collected Works of J. Richard Büchi, pp. 525–541. Springer (1990)
3. Church, A.: Logic, arithmetic, and automata. In: Proceedings of the international congress of mathematicians. vol. 1962, pp. 23–35 (1962)
4. Cimatti, A., Geatti, L., Gigante, N., Montanari, A., Tonetta, S.: Reactive synthesis from extended bounded response LTL specifications. In: 2020 Formal Methods in Computer Aided Design (FMCAD). pp. 83–92. IEEE (2020)
5. Cimatti, A., Geatti, L., Gigante, N., Montanari, A., Tonetta, S.: Expressiveness of Extended Bounded Response LTL. arXiv preprint arXiv:2109.08319 (2021)
6. Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In: 2012 Formal Methods in Computer-Aided Design (FMCAD). pp. 52–59. IEEE (2012)

7. Ehlers, R.: Symbolic bounded synthesis. In: International conference on Computer Aided Verification (CAV). pp. 365–379. Springer (2010)
8. Ehlers, R., Raman, V.: Slugs: Extensible GR(1) synthesis. In: International Conference on Computer Aided Verification. pp. 333–339. Springer (2016)
9. Faymonville, P., Finkbeiner, B., Rabe, M.N., Tentrup, L.: Encodings of bounded synthesis. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 354–370. Springer (2017)
10. Faymonville, P., Finkbeiner, B., Tentrup, L.: Bosy: An experimentation framework for bounded synthesis. In: International Conference on Computer Aided Verification (CAV). pp. 325–332. Springer (2017)
11. Filiot, E., Jin, N., Raskin, J.F.: An antichain algorithm for LTL realizability. In: International Conference on Computer Aided Verification (CAV). pp. 263–277. Springer (2009)
12. Finkbeiner, B., Hahn, C., Lukert, P., Stenger, M., Tentrup, L.: Synthesizing reactive systems from hyperproperties. In: International Conference on Computer Aided Verification (CAV). pp. 289–306. Springer (2018)
13. Finkbeiner, B., Schewe, S.: Bounded synthesis. International Journal on Software Tools for Technology Transfer **15**(5-6), 519–539 (2013)
14. Jacobs, S., Bloem, R.: The 5th reactive synthesis competition (SYNTCOMP 2018)
15. Jacobs, S., Bloem, R., Brenguier, R., Ehlers, R., Hell, T., Könighofer, R., Pérez, G.A., Raskin, J.F., Ryzhyk, L., Sankur, O., et al.: The first reactive synthesis competition (syntcomp 2014). International journal on software tools for technology transfer **19**(3), 367–390 (2017)
16. Kretínský, J., Meggendorfer, T., Sickert, S.: Owl: A library for ω-words, automata, and LTL. In: Lahiri, S.K., Wang, C. (eds.) Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11138, pp. 543–550. Springer (2018). https://doi.org/10.1007/978-3-030-01090-4_34, `https://doi.org/10.1007/978-3-030-01090-4_34`
17. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: 46th Annual Symposium on Foundations of Computer Science (FOCS). pp. 531–540. IEEE (2005)
18. Luttenberger, M., Meyer, P.J., Sickert, S.: Practical synthesis of reactive systems from LTL specifications via parity games. Acta Informatica **57**(1), 3–36 (2020)
19. Manna, Z., Pnueli, A.: A hierarchy of temporal properties (invited paper, 1989). In: Proceedings of the 9th annual ACM symposium on Principles of distributed computing. pp. 377–410 (1990)
20. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: International Conference on Computer Aided Verification. pp. 578–586. Springer (2018)
21. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive (1) designs. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 364–380. Springer (2006)
22. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: International Colloquium on Automata, Languages, and Programming (ICALP). pp. 652–671. Springer (1989)
23. Rosner, R.: Modular synthesis of reactive systems. Ph.D. thesis, PhD thesis, Weizmann Institute of Science (1992)
24. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and computation **115**(1), 1–37 (1994)
25. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: A symbolic approach to safety LTL synthesis. In: Haifa Verification Conference. pp. 147–162. Springer (2017)