GR(1)*: GR(1) specifications extended with existential guarantees

Gal Amram[✉], Shahar Maoz[✉], and Or Pistiner[✉]

Tel Aviv University, Tel Aviv, Israel

Abstract. Reactive synthesis is an automated procedure to obtain a correct-by-construction reactive system from its temporal logic specification. GR(1) is an expressive assume-guarantee fragment of LTL that enables efficient synthesis and has been recently used in different contexts and application domains. A common form of providing system’s requirements is through use cases, which are existential in nature. However, GR(1), as a fragment of LTL, is limited to universal properties. In this paper we introduce GR(1)*, which extends GR(1) with existential guarantees. We show that GR(1)* is strictly more expressive than GR(1) as it enables the expression of guarantees that are inexpressible in LTL. We solve the realizability problem for GR(1)* and present a symbolic strategy construction algorithm for GR(1)* specifications. Importantly, in comparison to GR(1), GR(1)* remains efficient: the time complexity of our realizability checking and synthesis procedures for GR(1)* is identical to the time complexity of the known corresponding procedures for GR(1).

Keywords: Reactive systems; Synthesis; GR(1); Existential guarantees

1. Introduction

Reactive synthesis is an automated procedure to obtain a correct-by-construction reactive system from its temporal logic specification [PR89]. Rather than manually constructing an implementation and using model checking to verify it against a specification, synthesis offers an approach where a correct implementation of the system is automatically obtained for a given specification, if such an implementation exists.

GR(1) is a fragment of linear temporal logic (LTL) [Pnu77], which has an efficient symbolic synthesis algorithm [BJP⁺12, PPS06] and whose expressive power covers most of the well-known LTL specification patterns of Dwyer et al. [DAC99, MR15a]. GR(1) specifications include assumptions and guarantees about what needs to hold on all initial states, on all states and transitions (safety), and infinitely often on every run (justice). GR(1) synthesis has been used and extended in different contexts and for different application domains, including robotics [KFP09, MSP⁺16, MR15b], scenario-based specifications [MS12], aspect languages [MS11], event-based behavior models [DBPU13], hybrid systems [FDL⁺16], and device drivers [WR14], to name a few.

Correspondence to: Shahar Maoz, e-mail: maoz@cs.tau.ac.il

A common form of providing system's requirements is through use cases [AM04, Jac04, Poh10]. In contrast to universal behaviors, i.e., which must hold on all possible system runs, use cases describe possible, existential behaviors. Use cases are commonly used in the early stages of requirements analysis and specification, as they are natural to define from a user's perspective and as in these stages, invariants may be too strong to be specified correctly. Use cases are useful also in specifying alternative and exceptional behaviors, which, by nature, do not appear in every run, and in specifying examples of behaviors that should not be possible. They are further commonly used again in later stages of development, to prescribe test cases. Despite all the above, to the best of our knowledge, no previous work has proposed efficient reactive synthesis for specifications that include not only universal but also existential properties.

In this work we present $\text{GR}(1)^*$, which extends $\text{GR}(1)$ specifications and synthesis with existential guarantees. $\text{GR}(1)^*$ allows engineers to naturally describe use cases as part of the specification and to efficiently synthesize a correct-by-construction controller that guarantees to make them possible.

We formally define $\text{GR}(1)^*$ and show that it is strictly more expressive than $\text{GR}(1)$ (as it can express properties that are in CTL^* [EH86] and outside LTL). We show how to solve $\text{GR}(1)^*$ games using a symbolic fixed-point algorithm, and present a corresponding controller construction.

Importantly, in comparison to $\text{GR}(1)$ [BJP⁺12], $\text{GR}(1)^*$ induces the same cost in terms of time complexity. Specifically, in [BJP⁺12], $\text{GR}(1)$ games are solved in time $O(nmN^2)$ where n is the number of justice guarantees, m is the number of justice assumptions, and N is the size of the state space, $2^{\mathcal{X} \cup \mathcal{Y}}$. The time complexity of our solution for $\text{GR}(1)^*$ games is $O((nm + \sum_{k=1}^{\ell} |reg_k|)N^2)$, where $reg_1, \dots, reg_{\ell}$ are the regular expressions that appear in the existential guarantees. Hence, both games are solved in time $O(|\varphi| N^2)$ where φ is the $\text{GR}(1)$ or $\text{GR}(1)^*$ formula.

A preliminary version of the present work has appeared in [AMP19]. In this journal paper we extend it as follows. First, we present a complete definition of $\text{GR}(1)^*$, which uses regular expressions. This complete definition generalizes the original restricted definition and makes it flexible and strictly more expressive. Second, we present another existential operator GE^+ (Sect. 7), a discussion and extensions for the case where the specification is well-separated (Sect. 8), and a discussion and results on unrealizability in $\text{GR}(1)^*$ games (Sect. 9). Third, we present an extended empirical evaluation, including an implementation of the proposed extended technique, and experiments performed over the well-known AMBA specification benchmark [ARM, BGJ⁺07], extended with existential guarantees (Sects. 2 and 10). Finally, we include proofs, which were not included in [AMP19].

The paper is organized as follows. Section 2 presents a motivating example, the well-known AMBA specification extended with existential guarantees. Section 3 presents required background definitions. In Sect. 4 we introduce $\text{GR}(1)^*$ formulas, and show that they extend the expressive power of the $\text{GR}(1)$ fragment of LTL. In Sect. 5 we show how to solve $\text{GR}(1)^*$ games using a symbolic fixed-point algorithm, and in Sect. 6 we present a corresponding controller construction. In Sect. 7 we present additional forms of existential guarantees, and corresponding ways to support them. In particular, we also show how to support past LTL formulas in existential guarantees. In Sect. 8 we prove that with well-separated environment specifications [KP10, MR16], our solution achieves a stronger winning condition, and in Sect. 8.2, we show that a well-separated environment enables support for deterministic Büchi properties in existential guarantees. In Sect. 9 we discuss $\text{GR}(1)^*$ unrealizability, and show that in contrast to $\text{GR}(1)$, some $\text{GR}(1)^*$ games admit states from which neither the system nor the environment has a winning strategy. Finally, in Sect. 10 we present an empirical evaluation of the proposed synthesis technique. Section 11 discusses related work and Sect. 12 concludes and suggests future research directions. All proofs are provided in Appx. A.

2. Example: ARM AMBA AHB arbiter

2.1. Background

ARM Advanced Microcontroller Bus Architecture (AMBA) is a widely acceptable industrial standard bus interface [ARM]. The most known bus defined within AMBA, Advanced High-performance Bus (AHB), constitutes a popular $\text{GR}(1)$ specification example and benchmark [BGJ⁺07, BJK14, CRST08, FMR20, GCH13, KHB13, KMR17, MRS19, MS20a, MS20b, PPS06]. The AHB arbiter handles data transfers from masters to slaves. Each master can raise a request to deliver data. The request can be an ordinary request or a locked-request.

When a locked-request is granted by the arbiter, the master performs a locked-access to the bus, which will not be interrupted until data-transfer is completed. There are three types of data-transfers: a single transfer, a four-length transfer, and an unspecified-length transfer.

We now present the input and output variables, with a description of their intuitive interpretation. For convenience, we also use finite-type variables that are not Boolean. Formally, these variables are to be replaced with several Boolean variables that model the value they store. In our description below, we specify the type of the variable only in case of a non-Boolean variable.

We start with the environment variables:

- HBUSREQ_i —Master i raises a request to access the bus.
- HLOCK_i —Master i raises a request to a locked-access to the bus. HLOCK_i is raised together with HBUSREQ_i .
- HBURST —Store values from the set $\{\text{SINGLE}, \text{BURST4}, \text{INCR}\}$, which indicate if the master requests to deliver a single transfer, a four-length transfer, or an unspecified-length transfer, respectively. When master i gets a permission to a locked-access to the bus, it announces the type of the transfer through variable HBURST .
- HREADY —Raised by a slave to indicate that data-processing is completed.

We continue with the system variables:

- HGRANT_i —The request by master i is granted, and master i can start and transfer data in the next step.
- HMASTER —Store values from $\{0, \dots, n\}$. For $i > 0$, $\text{HMASTER} = i$ means that master i currently owns the bus, and can deliver data. $\text{HMASTER} = 0$ indicates that no master is currently allowed to use the bus.
- HMASTLOCK —Raised when the master that currently owns the bus is performing a locked-access.
- START —Raised when a master starts to transfer data.
- LOCKED —Raised when some master performs a locked-access.
- DECIDE —Raised when the arbiter decides to grant a request by some master. Raising DECIDE enables changing the value of LOCKED , if the arbiter grants a locked-access, and the value of HGRANT_i , when master i 's request is to be granted.

We refer the reader to [BGJ⁺07] for the complete specification of AMBA AHB. The specification ensures that the following hold:

- The bus cannot be accessed by two masters simultaneously.
- Every request to access the bus will be granted eventually.
- A granted locked-access will not be interrupted.

The above can be written as a GR(1) specification.

2.2. Adding examples of existential guarantees

We now describe example existential guarantees one can add to the AMBA specification using GR(1)*. An existential guarantee is an expression of the form $\text{GEF}(\text{reg})$ where reg is a regular expression. The meaning of this expression is discussed and formally defined in Sect. 4.

First, assume that the requirements document describes a typical use case: master i requests a locked-access, and after a while its request is granted; while master i uses the bus, master j raises a request and is forced to wait at least until data-transfer by master i is completed. Thus, in our example, the engineer formalizes the following guarantees and adds them to the specification:

Use case $\mathbf{A}_{i,j}$. Master $j \neq i$ waits while master i performs a locked-access. $\text{GEF}(\text{reg})$ where $\text{reg} =$

$$\begin{aligned} & (\text{HBUSREQ}_i \wedge \text{HLOCK}_i \wedge \text{HMASTER} \neq i) \\ & (\text{HMASTER} \neq i)^* (\text{HMASTER} = i)^+ \\ & (\text{HBUSREQ}_j \wedge \text{HMASTER} = i) (\text{HMASTER} = i)^+ \text{HREADY}. \end{aligned}$$

Second, the engineer is aware that sometimes a synthesized controller may achieve its goals by preventing certain events from happening, and she wishes to avoid such vacuous solutions and problematic situations. As an example, she considers the allegedly possible situation, in which the arbiter always prevents unlocked-accesses from being completed. She thus formalizes and adds the following guarantees to the specification:

Use case B_i. Master i successfully completes an unlocked transfer. $GEF(reg)$ where $reg =$

$$(HMASTER = i \wedge \neg HMASTLOCK)^+ HREADY.$$

Finally, an example of using a negative scenario as a ‘test’. The engineer believes that the specified arbiter satisfies a strong fairness property: it is impossible that a master’s request remains pending while another master gets access to the bus twice.

To test this hypothesis, she checks whether the specification with the following additional existential guarantees is unrealizable:

Use case C_{i,j}. Master i raises a request, and master j gets to access the bus twice before granting master i ’s request. $GEF(reg)$ where $reg =$

$$\begin{aligned} & (HBUSREQ_i \wedge HMASTER \neq i \wedge HMASTER \neq j) \\ & (HMASTER \neq i)^*(HMASTER = j)(HMASTER \neq i)^* \\ & (HMASTER \neq i \wedge HMASTER \neq j)(HMASTER \neq i)^*(HMASTER = j). \end{aligned}$$

3. Preliminaries

In this section we present notations and definitions we use throughout the paper. We further survey some fundamental facts in the field that are important to establish our results. Our notations are standard and mostly based on [BJP⁺12].

3.1. Game structures and strategies

For a set of Boolean variables $\mathcal{V}$, a *state* is an element $s \in 2^{\mathcal{V}}$, an *assertion* is a Boolean formula over $\mathcal{V}$, and $\models$ is the satisfaction relation between a state and an assertion. `true` and `false` are the assertions satisfied by every state and by no state, resp. As an assertion naturally corresponds to the set of states by which it is satisfied, we refer to sets of states as assertions and we may write $s \models A$ instead of $s \in A$. For $\mathcal{Z} \subseteq \mathcal{V}$ and $s \in 2^{\mathcal{V}}$, $s|_{\mathcal{Z}}$ denotes the state $s \cap \mathcal{Z} \in 2^{\mathcal{Z}}$. For a set of variables $\mathcal{V}$, $\mathcal{V}'$ is the set of variables obtained by replacing each $v \in \mathcal{V}$ with v' . Likewise, for $s \in 2^{\mathcal{V}}$ and an assertion a over $\mathcal{V}$, $s' \in 2^{\mathcal{V}'}$ and a' are the state and assertion obtained by replacing each variable v with v' . If $\mathcal{V}_1, \dots, \mathcal{V}_k$ are pairwise disjoint sets of variables and $s_i \in 2^{\mathcal{V}_i}$, we write $(s_1, \dots, s_k)$ as an abbreviation for $s_1 \cup \dots \cup s_k$. Thus, $(s_1, \dots, s_k)$ is a state over $\mathcal{V} = \mathcal{V}_1 \cup \dots \cup \mathcal{V}_k$.

A *game structure* is a tuple $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, where $\mathcal{X}, \mathcal{Y}$ are disjoint sets of variables, θ^e is an assertion over $\mathcal{X}$, θ^s is an assertion over $\mathcal{X} \cup \mathcal{Y}$, ρ^e is an assertion over $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}'$, and ρ^s is an assertion over $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{Y}'$. Intuitively, a game structure defines how two players, the *environment* and the *system*, choose inputs and outputs repeatedly. θ^e and θ^s set rules for the beginning of the play; the environment chooses an initial input $s_x \models \theta^e$ and, in response, the system chooses an output s_y such that $(s_x, s_y) \models \theta^s$. Afterwards, the players take turns choosing inputs and outputs in compliance with the *safety assumptions and guarantees*, ρ^e and ρ^s , resp. Specifically, from a state $s \in 2^{\mathcal{X} \cup \mathcal{Y}}$, the environment can choose an input $s_x \in 2^{\mathcal{X}}$, such that $(s, s'_x) \models \rho^e$, and the system may respond with an output $s_y \in 2^{\mathcal{Y}}$ if $(s, s'_x, s'_y) \models \rho^s$.

A state s is said to be a *deadlock* for the system if there exists $s_x \in 2^{\mathcal{X}}$ such that $(s, s'_x) \models \rho^e$ and for all $s_y \in 2^{\mathcal{Y}}$, $(s, s'_x, s'_y) \not\models \rho^s$. Analogously, a deadlock for the environment is a state s for which there is no s_x such that $(s, s'_x) \models \rho^e$. A *play* is a sequence of states, $s_0, s_1, s_2, \dots$, such that (1) for two consecutive states s_i, s_{i+1} , $(s_i, s'_{i+1}) \models \rho^e \wedge \rho^s$, and (2) either it is infinite or it ends in a deadlock.

A *strategy* for the system from $S \subseteq 2^{\mathcal{X} \cup \mathcal{Y}}$ is a partial function $f^s : (2^{\mathcal{X} \cup \mathcal{Y}})^+ \times 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ such that (1) for $s_0 \in S$, (s_0) is consistent with f^s ; (2) if $(s_0, \dots, s_k)$ is consistent with f^s , s_k is not a deadlock for the system, and $(s_k, s'_x) \models \rho^e$ for $s_x \in 2^{\mathcal{X}}$, then $f^s(s_0, \dots, s_k, s_x)$ is defined, and for $s_y = f^s(s_0, \dots, s_k, s_x)$, $(s_k, s'_x, s'_y) \models \rho^s$, and the sequence $(s_0, \dots, s_k, (s_x, s_y))$ is consistent with f^s . We say that an infinite sequence of states is consistent with f^s if any of its finite prefixes is consistent with f^s . A strategy for the environment player is a partial function

$f^e : (2^{\mathcal{X} \cup \mathcal{Y}})^+ \rightarrow 2^{\mathcal{X}}$ that satisfies the analogous requirements. Consistency of a sequence with f^e is also defined analogously.

A controller determines a strategy from $S \subseteq 2^{\mathcal{X} \cup \mathcal{Y}}$ for the system using a finite memory. Formally, a controller C is a partial function with a set of memory values M . The controller has an initial value m_0 , and for some tuples $(s, s_x, m) \in 2^{\mathcal{X} \cup \mathcal{Y}} \times 2^{\mathcal{X}} \times M$ such that $(s, s'_x) \models \rho^e$, $C(s, s_x, m) = (s_y, \hat{m}) \in 2^{\mathcal{Y}} \times M$ such that $(s, s'_x, s'_y) \models \rho^s$. A controller C , from S , can also be viewed as a partial function over $(2^{\mathcal{X} \cup \mathcal{Y}})^+ \times 2^{\mathcal{X}}$. $C(s_0, \dots, s_k, s_x) = (s_y, m)$ if, from state s_0 , by receiving inputs $s_1 \upharpoonright_{\mathcal{X}}, \dots, s_k \upharpoonright_{\mathcal{X}}, s_x$, the controller replies with $s_1 \upharpoonright_{\mathcal{Y}}, \dots, s_k \upharpoonright_{\mathcal{Y}}, s_y$, and the final value of its memory is m . Formally, we require that the following holds:

- For $s_0 \in S$, (s_0) is consistent with C and for every $s_x \in 2^{\mathcal{X}}$ such that $(s_0, s'_x) \models \rho^e$, we require that $C(s_0, s_x, m_0)$ is defined, and if $C(s_0, s_x, m_0) = (s_y, m_1)$, we write $C(s_0, s_x) = (s_y, m_1)$;
- Assume that $(s_0, \dots, s_k)$ is consistent with C where $s_0 \in S$, and for $s_x \in 2^{\mathcal{X}}$, $C(s_0, \dots, s_k, s_x) = (s_y, m_{k+1})$. Then, $(s_0, \dots, s_k, (s_x, s_y))$ is consistent with C , and we require that for every $t_x \in 2^{\mathcal{X}}$ with $((s_x, s_y), t'_x) \models \rho^e$, $C((s_x, s_y), t_x, m_{k+1})$ is defined. Moreover, for $C((s_x, s_y), t_x, m_{k+1}) = (t_y, m_{k+2})$, we write $C(s_0, \dots, s_k, (s_x, s_y), t_x) = (t_y, m_{k+2})$.

Clearly, a controller C defines a strategy for the system f_C^s , by $f_C^s(s_0, \dots, s_k, s_x) = s_y$ iff $C(s_0, \dots, s_k, s_x) = (s_y, m)$ for some $m \in M$.

3.2. Linear temporal logic and the GR(1) fragment

Linear temporal logic (LTL) [Pnu77] is a language to specify properties over infinite words. Given a set of variables, $\mathcal{V}$, LTL formulas are generated by the grammar

$$\varphi ::= p \mid \neg \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi U \varphi,$$

where p is an assertion over $\mathcal{V}$ and parenthesis may be used to determine the order of operator activations. Given an infinite sequence of states $\pi \in (2^{\mathcal{V}})^\omega$, π^i denotes the suffix of π that starts from the i -th state in π (counting from zero). The term $\pi \models \varphi$ is defined inductively on the structure of φ : (1) $\pi \models p$ if $\pi(0) \models p$; (2) $\pi \models X\varphi$ if $\pi^1 \models \varphi$; (3) $\pi \models F\varphi$ if $\exists k (\pi^k \models \varphi)$; (4) $\pi \models G\varphi$ if $\forall k (\pi^k \models \varphi)$; (5) $\pi \models \varphi U \psi$ if $\exists k (\pi^k \models \psi \wedge \forall j < k (\pi^j \models \varphi))$; (6) Boolean operators are treated standardly.

Given a game structure $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, an LTL formula φ , and a play π , π wins for the system w.r.t. φ in GS if either it ends in a deadlock for the environment, or it is infinite and $\pi \models \varphi$. Otherwise, it wins for the environment w.r.t. φ in GS . A strategy for the system (resp. environment) f^s (resp. f^e) wins from $s \in 2^{\mathcal{X} \cup \mathcal{Y}}$ w.r.t. φ in GS if every play from s , consistent with f^s (resp. f^e), wins for the system (resp. environment) w.r.t. φ in GS . A strategy for the system (resp. environment) f^s (resp. f^e) wins from $S \subseteq 2^{\mathcal{X} \cup \mathcal{Y}}$ w.r.t. φ in GS if it wins from every $s \in S$. The winning region of the system (resp. environment) is the maximal set of states from which the system (resp. environment) wins. A strategy for the system f^s is a winning strategy w.r.t. φ in GS if for every $s_x \models \theta^e$ there exists $s_y \in 2^{\mathcal{Y}}$ such that $(s_x, s_y) \models \theta^s$ and f^s wins from (s_x, s_y) w.r.t. φ in GS . Analogously, a strategy for the environment f^e is a counter-strategy w.r.t. φ in GS if there exists $s_x \models \theta^e$ such that for every $s_y \in 2^{\mathcal{Y}}$ with $(s_x, s_y) \models \theta^s$, f^e wins from (s_x, s_y) w.r.t. φ in GS .

Among LTL formulas, of special interest to us is the GR(1) fragment [BJP⁺12]. A GR(1) formula is an LTL formula of the form $\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)$, where $a_1, \dots, a_m, g_1, \dots, g_n$ are assertions. The assertions $a_1, \dots, a_m$ are called *justice assumptions*, and $g_1, \dots, g_n$ are called *justice guarantees*.

3.3. μ -calculus over game structures

Modal μ -calculus [Koz82] is a modal logic enriched by least and greatest fixed-point (l.f.p. and g.f.p.) operators. Since we are interested in games that model reactive systems, we adopt the form of [BJP⁺12], which uses the controllable predecessor operators $\Diamond$ and $\Box$.

For a set of variables $\mathcal{V}$, and a set of relational variables $Var = \{X, Y, \dots\}$, a μ -calculus (in positive form) formula is constructed by the grammar

$$\phi ::= p \mid X \mid \phi \vee \phi \mid \phi \wedge \phi \mid \bigotimes \phi \mid \bigodot \phi \mid \mu X \phi \mid \nu X \phi,$$

where p is an assertion over $\mathcal{V}$. A μ -calculus formula defines a subset of $2^{\mathcal{V}}$. In words, $\bigotimes \phi$ defines the set of states from which the system can enforce reaching next a state in the set that ϕ defines, and $\bigodot \phi$ defines the set of states from which the environment can enforce reaching next a state in the set that ϕ defines. μ and ν denote the l.f.p. and g.f.p. operators, resp.

For a game structure, $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, and a valuation $\mathcal{E} : Var \rightarrow 2^{(2^{\mathcal{X} \cup \mathcal{Y}})}$, the semantics of a μ -calculus formula over $\mathcal{X} \cup \mathcal{Y}$ is defined by:

- For an assertion p , $\llbracket p \rrbracket_{GS}^{\mathcal{E}} = \{s \in 2^{\mathcal{V}} : s \models p\}$;
- For $X \in Var$, $\llbracket X \rrbracket_{GS}^{\mathcal{E}} = \mathcal{E}(X)$;
- $\llbracket \phi_1 \vee \phi_2 \rrbracket_{GS}^{\mathcal{E}} = \llbracket \phi_1 \rrbracket_{GS}^{\mathcal{E}} \cup \llbracket \phi_2 \rrbracket_{GS}^{\mathcal{E}}$; $\llbracket \phi_1 \wedge \phi_2 \rrbracket_{GS}^{\mathcal{E}} = \llbracket \phi_1 \rrbracket_{GS}^{\mathcal{E}} \cap \llbracket \phi_2 \rrbracket_{GS}^{\mathcal{E}}$;
- $\llbracket \bigotimes \phi \rrbracket_{GS}^{\mathcal{E}} = \left\{ s \in 2^{\mathcal{V}} : \begin{array}{l} \forall s_x \in 2^{\mathcal{X}} ((s, s'_x) \models \rho^e \Rightarrow \\ \exists s_y \in 2^{\mathcal{Y}} ((s, s'_x, s'_y) \models \rho^s \wedge (s_x, s_y) \in \llbracket \phi \rrbracket_{GS}^{\mathcal{E}})) \end{array} \right\}$;
- $\llbracket \bigodot \phi \rrbracket_{GS}^{\mathcal{E}} = \left\{ s \in 2^{\mathcal{V}} : \begin{array}{l} \exists s_x \in 2^{\mathcal{X}} ((s, s'_x) \models \rho^e \wedge \\ \forall s_y \in 2^{\mathcal{Y}} ((s, s'_x, s'_y) \models \rho^s \Rightarrow (s_x, s_y) \in \llbracket \phi \rrbracket_{GS}^{\mathcal{E}})) \end{array} \right\}$;
- $\llbracket \left\{ \begin{smallmatrix} \mu \\ \nu \end{smallmatrix} \right\} X \phi \rrbracket_{GS}^{\mathcal{E}} = \left\{ \bigcup_i S_i \right\}$, where $\left\{ \begin{smallmatrix} S_0 = \emptyset \\ S_0 = 2^{\mathcal{V}} \end{smallmatrix} \right\}$, $S_{i+1} = \llbracket \phi \rrbracket_{\mathcal{E}[X \leftarrow S_i]}^G$, and $\mathcal{E}[X \leftarrow S]$ denotes the valuation that is identical to $\mathcal{E}$ except that it maps X to S .

If all relational variables in ϕ are bound by fixed-point operators, we omit the notation $\mathcal{E}$, and just write $\llbracket \phi \rrbracket_{GS}$. We remark that by Knaster-Tarski theorem [Tar55], $\llbracket \mu X \phi \rrbracket_{GS}^{\mathcal{E}}$ and $\llbracket \nu X \phi \rrbracket_{GS}^{\mathcal{E}}$ indeed return the l.f.p. and g.f.p. of the function $S \mapsto \llbracket \phi \rrbracket_{GS}^{\mathcal{E}[X \leftarrow S]}$. This theorem can be applied since the positive form ensures that the function $S \mapsto \llbracket \phi \rrbracket_{GS}^{\mathcal{E}[X \leftarrow S]}$ is monotone, so l.f.p. and g.f.p. exist.

3.4. Finite automata

We survey few fundamental definitions and results from the field of automata theory. A *non-deterministic finite automaton* (NFA) is a tuple $\mathcal{A} = (Q, 2^{\mathcal{V}}, \Delta, q_0, \mathcal{F})$ where Q is a finite set of states, $\mathcal{V}$ is a set of variables, $\Delta : Q \times 2^{\mathcal{V}} \rightarrow 2^Q$ is the transition function, $q_0 \in Q$ is the initial state, and $\mathcal{F} \subseteq Q$ is the set of accepting states. For $q_i, q_j \in Q$ and $s \in 2^{\mathcal{V}}$, we write $q_i \xrightarrow{s} q_j$ as a shorthand for $q_j \in \Delta(q_i, s)$. We extend the notation $q_i \xrightarrow{s} q_j$ over words, inductively: (1) for every $q \in Q$, $q \xrightarrow{\epsilon} q$; (2) for $q_i, q_j \in Q$, and $w \in (2^{\mathcal{V}})^+$, $q_i \xrightarrow{w} q_j$, if $w = s\hat{w}$ where $s \in 2^{\mathcal{V}}$, and for some $q \in Q$, $q_i \xrightarrow{s} q$ and $q \xrightarrow{\hat{w}} q_j$. An NFA $\mathcal{A} = (Q, 2^{\mathcal{V}}, \Delta, q_0, \mathcal{F})$ *accepts* a word $w \in (2^{\mathcal{V}})^*$ if there exists $q \in \mathcal{F}$ such that $q_0 \xrightarrow{w} q$. The *language of the automaton*, $L(\mathcal{A}) \subseteq (2^{\mathcal{V}})^*$, is the set of all words accepted by the automaton $\mathcal{A}$.

An NFA $\mathcal{A} = (Q, 2^{\mathcal{V}}, \Delta, q_0, \mathcal{F})$ is a *deterministic finite automaton* (DFA) if for every $q \in Q$ and $s \in 2^{\mathcal{V}}$ $|\Delta(q, s)| = 1$. Standardly, from this point on, we shall prefer the notation δ to denote the transition function of a DFA, and we prefer Δ to denote the transition function of an NFA which is not deterministic. Moreover, for a DFA $\mathcal{A} = (Q, 2^{\mathcal{V}}, \delta, q_0, \mathcal{F})$, $q_i, q_j \in Q$, and $s \in 2^{\mathcal{V}}$, we write $\delta(q, s) = q_j$ instead of $\delta(q_i, s) = \{q_j\}$.

Let $\mathcal{V}$ be a set of states. A *regular expression* is generated by the grammar

$$reg ::= \emptyset \mid p \mid (reg) + (reg) \mid (reg)(reg) \mid (reg)^*,$$

where p is an assertion. The semantics of a regular expression reg is a language, $L(reg) \subseteq (2^{\mathcal{V}})^*$, defined by:

- For $reg = \emptyset$, $L(reg) = \emptyset$.
- For $reg = p$, $L(reg) = \{s \in 2^{\mathcal{V}} : s \models p\}$.
- For $reg = (reg_1) + (reg_2)$, $L(reg) = L(reg_1) \cup L(reg_2)$.

- For $reg = (reg_1)(reg_2)$, $L(reg) = L(reg_1)L(reg_2)$, where $w \in L(reg_1)L(reg_2)$ if w can be written as $w = w_1w_2$ such that $w_1 \in L(reg_1)$ and $w_2 \in L(reg_2)$.
- For $reg = (reg_1)^*$, $L(reg) = L(reg_1)^*$, where $w \in L(reg_1)^*$ if w can be written as $w = w_1w_2 \cdots w_k$ such that $k \in \mathbb{N}$ and each $w_i \in L(reg_1)$. Note the when $k = 0$, $w_1 \cdots w_k = \varepsilon$.

A fundamental result states that a language $L \subseteq (2^\mathcal{V})^*$ is accepted by some NFA iff it is accepted by some DFA iff it is the language of some regular expression [Hop08].

A *non-deterministic Büchi automaton* (NBW) accepts infinite words. It is defined in the same way as an NFA, and differs from a NFA in its acceptance condition. An NBW $\mathcal{A} = (Q, 2^\mathcal{V}, \Delta, q_0, \mathcal{F})$ *accepts* a word $w \in (2^\mathcal{V})^\omega$ if w can be written as a concatenation of non-empty finite words $w = w_1w_2 \cdots$ such that there exists a sequence of accepting states $q_1, q_2, \dots \in \mathcal{F}^\omega$ that satisfies: $\forall i \in \mathbb{N} (q_i \xrightarrow{w_i} q_{i+1})$. The language of the automaton $L(\mathcal{A}) \subseteq (2^\mathcal{V})^\omega$ is the set $\{w \in (2^\mathcal{V})^\omega : \mathcal{A} \text{ accepts } w\}$. Thus, the language of an NBW is an ω -language i.e. a language of infinite words. An NBW is a *deterministic Büchi automaton* (DBW) if for every $q \in Q$ and $s \in 2^\mathcal{V}$, $|\Delta(q, s)| = 1$. As for finite automata, we shall use the notation δ to denote the transition function of a DBW, and write $\delta(q_i, s) = q_j$ instead of $\delta(q_i, s) = \{q_j\}$.

An ω -regular expression is an expression of the form $reg_1 \widehat{reg_1}^\omega + \cdots + reg_n \widehat{reg_n}^\omega$ where $reg_1, \widehat{reg_1}, \dots, reg_n, \widehat{reg_n}$ are regular expressions, and for each $i \in \{1, \dots, n\}$, $\varepsilon \notin L(\widehat{reg_i})$. The language of the ω -regular expression $reg_1 \widehat{reg_1}^\omega + \cdots + reg_n \widehat{reg_n}^\omega$ is an ω -language defined by $L(reg) = \bigcup_{i=1}^n L(reg_i \widehat{reg_i}^\omega)$, where $w \in L(reg_i \widehat{reg_i}^\omega)$ if w can be written as $w = w_0w_1 \cdots$ such that $w_0 \in L(reg_i)$, and $\forall j > 0 (w_j \in L(\widehat{reg_i}))$. Such a language is said to be an ω -regular language. An ω -language is an ω -regular language iff it is the language of some NBW. But, some ω -regular languages are not recognized by any DBW [GTW02].

4. GR(1)*: going beyond LTL

We are now ready to define GR(1)* formulas and winning conditions. Following the definition, we first show that GR(1)* formulas capture properties that are inexpressible in LTL. In fact, already the restricted form of GR(1)* that we presented in the preliminary version of this paper [AMP19], allows one to express properties that are expressible neither in LTL nor in CTL [CE81]. We then show that the more general GR(1)* we define here captures properties that are inexpressible in CTL*.

We further prove that in the context of game-structures, a GR(1)* winning condition cannot be expressed via an LTL formula. Consequently, in particular, this implies that GR(1)* winning conditions are more expressive than GR(1) winning conditions.

4.1. GR(1)* formulas

GR(1)* extends the GR(1) fragment of LTL with existential guarantees of the form $GEF(reg)$, where reg is a regular expression over a set of variables $\mathcal{V}$. As in the case of justice guarantees, these existential guarantees should hold if all justice assumptions are satisfied infinitely often. Therefore, since this formula prescribes a possible behavior, the synthesized controller should either make it possible to satisfy reg infinitely often, or enforce the violation of at least one of the assumptions.

Definition 4.1 (GR(1)*) A GR(1)* formula over a set of variables $\mathcal{V}$ is a formula of the form

$$A \left(\bigwedge_{j=1}^m GF(a_j) \rightarrow \left(\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k) \right) \right),$$

where $a_1, \dots, a_m, g_1, \dots, g_n$ are assertions over $\mathcal{V}$, and $reg_1, \dots, reg_\ell$ are regular expressions over $\mathcal{V}$.

GR(1)* uses CTL* operators, together with regular expressions. We treat regular expressions as atomic path formulas, with the natural semantics: $\pi = s_0, s_1, \dots \models reg$, if $\exists k \geq 0 ((s_0, \dots, s_k) \in L(reg))$. Note that our definition includes existential guarantees but no existential assumptions. Adding such assumptions would only make it easier for the system to win by enforcing violation of the assumptions, which is undesirable. Still, importantly, note that the existential guarantees in GR(1)* may use not only system variables but also environment variables.

Fig. 1. Relationships between $GR(1)^*$, CTL^* , and subsets of CTL^*

In [AMP19], we presented a restricted form of $GR(1)^*$, in which the existential guarantees are of the form $GEF(e_1 \wedge F(e_2 \wedge F(e_3 \cdots F(e_r) \cdots)))$, where $e_1, \dots, e_r$ are assertions. This is, effectively, a restricted form of $GR(1)^*$ since such an existential guarantee can be replaced with $GEF(reg)$ where reg is a regular expression whose language is that of the following NFA:

In words, for $i < j$, $q_i \xrightarrow{e_{i+1} \wedge \dots \wedge e_j} q_j$, and for $i > 0$, $q_i \xrightarrow{\text{true}} q_i$. For the scope of this section, we denote this restricted fragment of $GR(1)^*$ from [AMP19] by $GR(1)^*[\text{rest}]$.

In [AMP19], we showed that both syntactically and semantically, $GR(1)^*[\text{rest}]$ is a fragment of CTL^* that is neither a subset of LTL nor a subset of CTL. For instance, the $GR(1)^*[\text{rest}]$ formula $A(GF(a) \rightarrow (GF(g) \wedge GEF(e)))$ is expressible neither in LTL nor in CTL (this can be proved using arguments similar to [EH86]). Moreover, the $GR(1)^*$ formula $A(GF(\text{true}) \rightarrow (GF(\text{true}) \wedge GEF((ab)^*c)))$ is inexpressible in CTL^* [MMDdJ11]. All the above establish the relationships depicted in Fig. 1.

4.2. $GR(1)^*$ winning condition

We now define when a strategy is winning w.r.t. a $GR(1)^*$ winning condition. As $GR(1)^*$ uses CTL^* operators, we essentially apply the definition of winning strategies for reactive systems with CTL^* winning conditions [KV00] (while treating regular expressions as path formulas, as explained above). That is, roughly, we wish to say that a strategy is winning if it induces a computation tree that satisfies the $GR(1)^*$ formula. However, since, in contrast to [KV00], we consider game structures that restrict the steps that the players can perform, we cannot directly apply the definition of [KV00], and some technical changes are necessary. Specifically, a strategy may induce a tree that has a branch that is finite (rather than infinite) and ends in a deadlock. Thus, this tree is not a computation tree in the sense of [KV00], and our formal definition takes this fact into consideration.

Consider a game structure $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, and a $GR(1)^*$ formula $\psi = A(\bigwedge_{j=1}^m GF(a_j) \rightarrow (\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k)))$ over $\mathcal{X} \cup \mathcal{Y}$.

Definition 4.2 Let f^s be a strategy for the system, and $\pi = \pi(0), \pi(1), \dots$ a play in GS , consistent with f^s . The pair (f^s, π) wins for the system w.r.t. ψ in GS if one of the following holds:

1. π is finite and it reaches a deadlock for the environment.
2. π is infinite and $\pi \models FG(\neg a_j)$ for some justice assumption a_j .
3. π is infinite, and the following hold:

$$\bigwedge_{i=1}^n GF(g_i): \pi \models \bigwedge_{i=1}^n GF(g_i).$$

$$\bigwedge_{k=1}^\ell GEF(reg_k): \text{For every } i \geq 0, \text{ and an existential guarantee } GEF(reg_k), \text{ the } i\text{th prefix of } \pi, \pi(0), \dots, \pi(i), \\ \text{can be extended to a play } \tilde{\pi}, \text{ consistent with } f^s, \text{ such that for some } r_0 \text{ and } r_1, \text{ with } i \leq r_0 \leq r_1, \text{ it holds} \\ \text{that } \tilde{\pi}(r_0) \dots \tilde{\pi}(r_1) \in L(reg_k).$$

Definition 4.3 (GR(1)* winning condition) A strategy for the system f^s wins from a state s w.r.t. ψ in GS , if for every play π from s , consistent with f^s , the pair (f^s, π) wins for the system w.r.t. ψ in GS . f^s wins from a set of states S w.r.t. ψ in GS if it wins from every $s \in S$. Further, f^s is winning for the system w.r.t. ψ in GS if for every $s_x \models \theta^e$ there exists s_y such that $(s_x, s_y) \models \theta^s$ and f^s wins for the system from (s_x, s_y) w.r.t. ψ in GS .

Throughout the paper, when the GR(1)* formula or the game structure we discuss are clear from the context, we omit their name and write, for example, “ f^s is a winning strategy” instead of “ f^s is a winning strategy w.r.t. ψ in GS ”.

4.3. Inexpressibility of GR(1)* winning conditions in LTL

The fact that GR(1)* is inexpressible in LTL (see Sect. 4.1), does not imply that a GR(1)* winning condition cannot always be replaced with an LTL winning condition. To conclude this form of inexpressibility, we show that the arguments of [EH86] apply in the context of synthesized reactive systems.

We say that an LTL winning condition φ is equivalent to a GR(1)* winning condition ψ , if for any game structure GS , and for any strategy for the system f^s , f^s is winning w.r.t. φ in GS , iff it is winning w.r.t. ψ in GS . Unsurprisingly, the example of [EH86] works in our context as well, and proves that GR(1)* winning conditions are inexpressible in LTL.

Proposition 4.4 GR(1)* winning conditions are inexpressible in LTL.

The proof of Prop. 4.4 reveals that even a winning condition as simple as $A(GEF(y))$ is inexpressible in LTL. Using similar arguments one can show that every existential guarantee from our motivating example (Sect. 2) is inexpressible in LTL.

Remark 4.5 Note that we consider equivalence only in terms of the winning condition. This does not mean that GR(1)* winning conditions cannot be expressed by other formulas, while imposing more radical changes in the game structure. In the context of reactive synthesis, such seemingly unexpected reductions exist. For example, CTL* formulas can be expressed by LTL winning conditions, but the encoding is involved with the addition of an exponential number of variables [BSK17]. Similarly, using the technique of [MR15a] (which we also use in Sect. 7), GR(1)* can be expressed by GR(1)*[rest] where each regular expression reg adds $\log |reg|$ variables to the game structure. We do not know if GR(1)* can be reduced to LTL or GR(1) while changing also the game structure, or what is the expected cost of such a reduction, if one exists.

5. Solving GR(1)* games

In this section we present a μ -calculus formula that computes the winning region of the system player in GR(1)* games. Consider a game structure $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, together with a GR(1)* formula $A(\bigwedge_{j=1}^m GF(a_j) \rightarrow (\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k)))$. To compute the system’s winning region, we present a μ -calculus formula that consists of three components: V , $g(Z)$, and $\{h_k(Z): k = 1, \dots, \ell\}$, each of which we define next.

First, note that enforcing a violation of the assumptions, if possible, ensures winning. The states from which the system can violate the assumptions are those from which it can win the game whose winning condition is the LTL formula $\bigvee_{j=1}^m FG(\neg a_j)$. By [KPP05, Lem. 9], these states are characterized by the μ -calculus formula:

$$\mu Y \left(\bigvee_{j=1}^m \nu X (\otimes Y \vee (\neg a_j \wedge \otimes X)) \right) \quad (1)$$

Thus, the first component we consider is $V = \llbracket \mu Y (\bigvee_{j=1}^m \nu X (\otimes Y \vee (\neg a_j \wedge \otimes X))) \rrbracket_{GS}$, the set of states computed by the μ -calculus formula in Eq. 1.

The second component we consider is the formula from [BJP⁺12] for solving GR(1) games. The justice assumptions and guarantees part of our GR(1)* formula, $\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)$, is solved by the formula in Eq. 2:

$$\nu Z \left(\bigwedge_{i=1}^n \mu Y \left(\bigvee_{j=1}^m \nu X ((g_i \wedge \otimes Z) \vee \otimes Y \vee (\neg a_j \wedge \otimes X)) \right) \right) = \nu Z(g(Z)) \quad (2)$$

For the third component, we turn to look at the regular expressions that appear in the existential guarantees of the GR(1)* formula, $\{reg_k : 1 \leq k \leq \ell\}$. For a regular expression reg_k and a set of states Z , we are interested in the set of states from which there exists a path in Z that satisfies $F(reg_k)$. Equivalently, we consider the set of all states $s_0 \in Z$ for which there exists a path of states in Z , $(s_0, \dots, s_r) \in L(\text{true}^* reg_k)$, such that $(s_i, s_{i+1}) \models \rho^s \wedge \rho^e$ for each two consecutive states in that sequence.

To compute this set of states we look at the graph obtained by the product of the game structure with an automaton for $L(\text{true}^* reg_k)$, and identify the states from which there is a path that leads to an accepting state of the automaton. For the construction, for each regular expression reg_k that appears in the formula, we fix an NFA $\mathcal{A}_k = (Q^k, 2^{\mathcal{X} \cup \mathcal{Y}}, \Delta^k, q_0^k, \mathcal{F}^k)$, with $L(\mathcal{A}_k) = L(\text{true}^* reg_k)$. The vertices of the graph we define next are of the form (q, s) where $q \in Q^k$ and s is a state. Intuitively, the state (q, s) indicates that the automaton has reached state q , after reading the symbol s .

Definition 5.1 $G_{(k,Z)} = (V_{(k,Z)}, E_{(k,Z)})$ is the graph defined by:

- $V_{(k,Z)} = Q^k \times Z$.
- $((q, s), (\hat{q}, \hat{s})) \in E_{(k,Z)}$ if $(s, \hat{s}) \models \rho^e \wedge \rho^s$, and $\hat{q} \in \Delta^k(q, \hat{s})$.

$R_{(k,Z)}(\mathcal{F}^k \times Z)$ is the set of all states in $V_{(k,Z)}$ from which there is a path in $G_{(k,Z)}$ to a state of $\mathcal{F}^k \times Z \subseteq V_{(k,Z)}$.

Thus, the third component we consider is:

$$h_k(Z) = \{s \in Z : \exists q \in Q^k ((q, s) \in R_{(k,Z)}(\mathcal{F}^k \times Z) \text{ and } q \in \Delta^k(q_0^k, s))\} \quad (3)$$

The graph $G_{(k,Z)}$ has $|Q^k| \times |Z|$ states. $\mathcal{A}_k$ can be constructed so that $|Q^k| \leq 2|reg_k| + 1$. Therefore, the set $h_k(Z)$ can be computed in $O(|reg_k| \cdot |Z|) \leq O(|reg_k| \cdot N)$ time. Note that this time complexity is achieved as we construct an NFA $\mathcal{A}_k$ from reg_k , a construction performed in linear time. The fact that we do not need to determinize $\mathcal{A}_k$ enables the efficiency of our synthesis technique.

Finally, combining the three components from Eq. 1-3, we obtain the formula in Eq. 4, which computes the winning region of the system in the GR(1)* game:

$$\nu Z (V \vee (g(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z))) \quad (4)$$

Theorem 5.2 (Realizability) $W = \llbracket \nu Z (V \vee (g(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z))) \rrbracket_{GS}$ is the winning region of the system player in the GR(1)* game.

In the next section, we show how to construct a winning strategy from the set $\llbracket \nu Z (V \vee (g(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z))) \rrbracket_{GS}$. Hence, to conclude the correctness of Thm. 5.2, we need to show that the system cannot win from every state in the complementary set of $\llbracket \nu Z (V \vee (g(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z))) \rrbracket_{GS}$.

Lemma 5.3 If $s \notin \llbracket \nu Z(V \vee (g(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z))) \rrbracket_{GS}$, then no strategy for the system wins from s .

By Thm. 5.2, a naive approach provides a realizability check for GR(1)* games in $O(nmN^3 + (\sum_{k=1}^{\ell} |reg_k| + N^2))$ symbolic steps. As in the case of GR(1), with the technique proposed in [BCJ⁺97], this can be improved to $O((nm + \sum_{k=1}^{\ell} |reg_k|)N^2)$ steps. We see that the time complexity of the realizability check for GR(1)* formulas is the same as the time complexity of the realizability check for GR(1) formulas. Specifically, both are solved in time $O((|\varphi| + N)^2)$, where φ is the GR(1) or GR(1)* formula.

6. Strategy construction

In this section we present a construction for a GR(1)* controller, which wins the GR(1)* game from the system's winning region W , computed by the formula in Eq. 4. The basic idea behind the construction is simple, and we start by describing it along with an informal description of a GR(1)* controller construction. Then, we point out that this simple construction has a significant disadvantage. We therefore suggest a way to improve the construction and present a detailed construction for an improved GR(1)* controller.

6.1. Construction discussion and overview

The basic idea of the construction is to alternate between two phases. Phase I is the well-known GR(1) strategy from [BJP⁺12]. The GR(1) controller satisfies the justice guarantees $g_1, \dots, g_n$, one by one, provided that it will fail to satisfy g_i only in case the justice assumptions are violated forever. We add to this phase the requirement that if the controller reaches a state in V , it forces violation of the justice assumptions for the remainder of the play. After satisfying the justice guarantees, the GR(1)* controller proceeds to phase II in which it chooses a regular expression reg_k , and tries, cooperatively, to satisfy $F(reg_k)$.

Although correct, the strategy outlined above suffers from a significant drawback: it forces the violation of the assumptions whenever possible, although in some cases the system can win the game while giving the environment an opportunity to satisfy its assumptions. In the context of GR(1) games, some works suggested to settle this issue by allowing the satisfaction of the assumptions in cases where the justice guarantees can be satisfied as well [BEK15, EKB15, MPS19]. In our context of GR(1)*, this approach should be taken carefully, since it is possible that such a strategy will not allow the satisfaction of the existential guarantees. Specifically, by taking this approach, a play can be led into a state from which there is no path that satisfies $F(reg_k)$ for some regular expression reg_k .

To improve the described GR(1)* controller by avoiding unnecessary coercion of justice assumption violation, we note that the existential guarantees can be satisfied from the set $\bigcap_{k=1}^{\ell} h_k(W)$. Hence, we refer to $V' = W \setminus \bigcap_{k=1}^{\ell} h_k(W)$ as the “unsafe” region, from which we must force violation of the assumptions. The following Lemma claims that, indeed, the system can force violation of the assumptions from V' .

Lemma 6.1 $V' \subseteq V$.

Therefore, we improve the GR(1)* construction described above as follows: in phase I we satisfy the justice guarantees one by one, and in phase II we try to satisfy, cooperatively, an existential guarantee. Further, if the play reaches V' while in phase I, we turn to force violation of the justice assumptions for the remainder of the play.

6.2. Detailed construction

We now present the construction in detail. Our GR(1)* controller activates several “sub-controllers”, which we describe below: C_V and $C_{(GR(1),1)}, \dots, C_{(GR(1),n)}$ are activated in phase I; $C_1, \dots, C_k$ are activated in phase II.

C_V . C_V is a memoryless controller that forces from V the violation of the assumptions, i.e., it wins from V the game whose winning condition is $\bigvee_{j=1}^m FG(\neg a_j)$. Clearly, $\bigvee_{j=1}^m FG(\neg a_j) \equiv (\bigvee_{j=1}^m FG(\neg a_j)) \vee F(\text{false} \wedge \odot V)$. Hence, we apply the results of [KPP05, Lem. 9] and construct C_V using the technique proposed in [BJP⁺12].

$C_{(\text{GR}(1),i)}$. For each $1 \leq i \leq n$, $C_{(\text{GR}(1),i)}$ is a memoryless controller that forces from $\llbracket g(W) \rrbracket_{GS}$ (see Eq. 2) the satisfaction of the formula $(\bigvee_{j=1}^m FG(\neg a_j)) \vee F(g_i \wedge \bigotimes W)$. The construction of these controllers is described in [BJP⁺12]. We construct $C_{(\text{GR}(1),i)}$ so that it keeps taking steps also after reaching its goal. That is, if $s \in \llbracket g(W) \rrbracket_{GS}$, $s \models g_i \wedge \bigotimes W$, and $(s, s'_x) \models \rho^e$, then $C_{(\text{GR}(1),i)}(s, s_x) = s_y$ such that $(s, s'_x, s'_y) \models \rho^s$ and $(s_x, s_y) \in W$.

C_k . C_k is a $|Q^k|$ -memory controller (see Def. 5.1 and the discussion that precedes it) that, from each $s \in h_k(W)$, tries to walk on a path in W that satisfies $F(\text{reg}_k)$. Roughly speaking, the controller's memory variable *mem* stores the current state of the automaton. Hence, if $C_k(s, s_x, i) = (s_y, j)$, then $q_j^k \in \Delta^k(q_i^k, (s_x, s_y))$ (or, equivalently, $((q_i^k, s), (q_j^k, (s_x, s_y))) \in E_{(k,W)}$). For convenience, we assume that $Q^k = \{q_0^k, q_1^k, \dots, q_{|Q^k|-1}^k\}$ thus *mem* stores values from $\{0, \dots, |Q^k| - 1\}$. Possibly, for some states $(s, s_x) \in W \times 2^X$, the controller has no defined response. We assume that in those cases the controller returns a designated “undefined” value. We present the construction in Alg. 1. The operator $\Diamond$ is used in Alg. 1 to denote the set of all predecessors of a set of vertices. That is, for a set of vertices X of graph $G = (V, E)$, $\Diamond X = \{v \in V : \exists u \in X((v, u) \in E)\}$.

Algorithm 1 The controller C_k

```

1:  $X \leftarrow \{(q_i^k, s) \in V_{(k,W)} : q_i^k \in \mathcal{F}^k\}$ 
2:  $X' \leftarrow \Diamond X \setminus X$ 
3: while  $X' \neq \emptyset$  do
4:   for all  $(q_j^k, s) \in X'$  and  $s_x \in 2^X$  such that  $(s, s'_x) \models \rho^e$  do
5:     if there exists  $(q_i^k, s_y) \in Q^k \times 2^Y$  such that
6:       •  $(q_i^k, (s_x, s_y)) \in X$ , and
7:       •  $((q_j^k, s), (q_i^k, (s_x, s_y))) \in E_{(k,W)}$  then
8:         choose  $(q_i^k, s_y)$  as above
9:          $C_k(s, s_x, j) \leftarrow (s_y, i)$ 
10:      end if
11:   end for
12:    $X \leftarrow X \cup X'$ 
13:    $X' \leftarrow \Diamond X \setminus X$ 
14: end while
```

Algorithm 2 presents the GR(1)* controller's construction. For clarity of presentation, we present the GR(1)* controller via a simple pseudocode. The translation of the code into a construction of a symbolic controller using BDD [Bry86] operations is straightforward.

The GR(1)* controller's memory consists of several variables. *next_jus* specifies the number of the next justice guarantee the controller strives to fulfill, and thus stores values from $\{1, \dots, n\}$. *next_ex* specifies the number of the ensuing existential guarantee that we aim to satisfy, and thus stores values from $\{1, \dots, \ell\}$. *mem* is a variable used by the controllers $C_1, \dots, C_\ell$. Hence, it stores values from $\{0, \dots, |Q^{k'}| - 1\}$, where $|Q^{k'}| = \max\{|Q^k| : 1 \leq k \leq \ell\}$. *phase* indicates which phase we now aim to. When *phase* = 1, we want to satisfy the justice guarantee $g_{\text{next_jus}}$ (phase I), and when *phase* = 2, we are trying, cooperatively, to satisfy the existential guarantee $\text{reg}_{\text{next_ex}}$ (phase II). In line 19, $\oplus 1$ means that we write the next value to *next_ex*. That is, for $k < \ell$, $k \oplus 1 = k + 1$, and $\ell \oplus 1 = 1$.

In phase I, first, the GR(1)* controller checks if the play reached a V' -state, and if so, the GR(1)* controller activates C_V for the remainder of the play and thus forces violation of the justice assumptions (line 2). Otherwise, the GR(1)* controller activates $C_{(\text{GR}(1), \text{next_jus})}$. The controller $C_{(\text{GR}(1), \text{next_jus})}$ either violates the justice assumptions, or reaches a state $\hat{s} \models g_{\text{next_jus}} \wedge \bigotimes W$. When the latter occurs, the GR(1)* controller increases *next_jus* by 1 (line 5), unless *next_jus* stores n . In the case where *next_jus* stores n (line 7), the GR(1)* controller checks if the existential guarantee *next_ex* can be satisfied (line 8). If the condition in line 8 holds, the GR(1)* controller initializes *mem* and switches to phase II: the next state in that play is (s_x, s_y) . Thus, the GR(1)* controller writes a value i to *mem*, such that $q_0^{\text{next_ex}} \xrightarrow{(s_x, s_y)} q_i^{\text{next_ex}}$ (line 9), and it writes 2 to *phase* (line 10). Note that if the

condition in line 8 does not hold, then the ensuing state $(s_x, s_y) \notin h_{next_ex}(W)$ and thus $(s_x, s_y) \in V'$. In this case, the GR(1)* controller does not perform a phase-switch, and it will activate C_V for the remainder of the play.

Finally, if *phase* stores 2 (line 15), the GR(1)* controller activates C_{next_ex} . This controller tries, with the cooperation of the environment, to satisfy the existential guarantee $F(reg_{next_ex})$. Once $F(reg_{next_ex})$ has been satisfied, or once the GR(1)* controller receives an input that is not in correspondence with the path that the controller C_{next_ex} is trying to walk on, the GR(1)* controller switches back to phase I (line 19). In addition, it writes 1 to *next_jus* to satisfy, again, all of the justice guarantees one by one, and replaces *next_ex* with $next_ex \oplus 1$.

Algorithm 2 The GR(1)* controller

```

/*      Initialization      */
input:  $s_x \models \theta^e$ 
1:  $(next\_jus, next\_ex, mem, phase) \leftarrow (1, 1, 1, 1)$ 
2: return  $s_y$ , such that  $(s_x, s_y) \models (\theta^s \wedge W)$ 

/*      From state  $s$       */
input:  $s_x$  such that  $(s, s'_x) \models \rho^e$ 
1: if phase == 1 then                                     // Phase I: Fulfilling a justice guarantee
2:   if  $s \models V'$  then return  $C_V(s, s_x)$                   // The play reached  $V'$ 
3:   end if
4:    $s_y \leftarrow C_{(GR(1), next\_jus)}(s, s_x)$ 
5:   if  $((s_x, s_y) \models g_{next\_jus} \wedge \bigotimes W) \wedge (next\_jus < n)$  then  $next\_jus \leftarrow next\_jus + 1$ 
6:   end if
7:   if  $((s_x, s_y) \models g_{next\_jus} \wedge \bigotimes W) \wedge (next\_jus = n)$  then
8:     if there exists  $i$  such that
       •  $(q_i^{next\_ex}, (s_x, s_y)) \in R_{(next\_ex, W)}(\mathcal{F}^k \times W)$ , and
       •  $q_i^{next\_ex} \in \Delta^{next\_ex}(q_0, (s_x, s_y))$  then
9:        $mem \leftarrow i$  as above
10:       $phase \leftarrow 2$ 
11:    end if
12:  end if
13:  return  $s_y$ 
14: end if
15: if phase == 2 then                                     // Phase II: Fulfilling an existential guarantee
16:    $(s_y, mem) \leftarrow C_{next\_ex}(s, s_x, mem)$ 
17:   if  $s_y \neq \text{"undefined"}$  then return  $s_y$ 
18:   end if
19:    $(next\_jus, next\_ex, phase) \leftarrow (1, next\_ex \oplus 1, 1)$ 
20:   goto line 1
21: end if

```

The reader may notice few seemingly problematic issues concerning the construction that require clarification: (1) once the play reaches V' , it must not leave V' ; (2) the controllers $C_{(GR(1), 1)}, \dots, C_{(GR(1), n)}$ operate from $\llbracket g(W) \rrbracket_{GS}$, but the initialization in Alg. 2 returns a state in W ; (3) similarly, after reaching a state $s \models g_i \wedge \bigotimes W$, we want to activate $C_{(GR(1), i+1)}$, which operate from $\llbracket g(W) \rrbracket_{GS}$; (4) phase I ends by reaching a state $s \models g_n \wedge \bigotimes W$, from which we want to activate some C'_k , which operates from $h_k(W)$. The correctness proof we provide in the appendix settles these issues as follows: (1) we show that if $s \in V'$ and $(s, s'_x) \models \rho^e$, then $(s_x, C_V(s, s_x)) \in V'$; (2,3) we show that $W \cup \bigotimes W \subseteq \llbracket g(W) \rrbracket_{GS}$; (4) we show that $\bigotimes W \subseteq W$ and thus,¹ if $s \models \bigotimes W$, then either $s \in V'$ and C_V can be activated, or $s \in \bigcap_{k=1}^{\ell} h_k(W)$, and each C_k can be activated.

Theorem 6.2 (Controller construction) The controller C as constructed according to Alg. 2 implements a winning strategy for the system from W with a memory of size $2 \cdot n \cdot \ell \cdot (\max_{1 \leq k \leq \ell} |Q_k|)$.

¹In fact, $W = \bigotimes W$, but we do not use this observation in the correctness proof of the algorithm.

As a final remark, note that in lines 5 and 7, the condition $(s_x, s_y) \models g_{next_jus} \wedge \bigotimes W$ can be replaced with $(s_x, s_y) \models g_{next_jus}$, for optimization purposes. We now explain the correctness of this replacement. In the proof of Thm. 6.2, we show that $\bigotimes W \subseteq W$. Moreover, Thm. 6.2 and Lem. 5.3 ensure that a play that is consistent with the GR(1) controller, as constructed in Alg. 2, never reaches a deadlock for the system, and never leaves W . Hence, for every state s that is reachable by the GR(1) controller, $s \models \bigotimes W$ and there is no reason to check this condition.

7. The GE^+ operator

In this section we motivate and define another existential operator for $GR(1)^*$, GE^+ , which relates to satisfaction from every prefix rather than from every reachable state. We then show that the new operator is useful in expressing several different kinds of existential requirements, including, e.g., ones involving past LTL operators.

7.1. Motivating and defining the GE^+ operator

Consider again the example in Sect. 2, together with another use case:

Use case $D_{i,j}$. Master i raises a request and, although master j raises a request afterwards and master i does not lower its request, master j gets to access the bus before master i .

This use case is formalized by the regular expression $reg = \text{true}^*(\text{HBUSREQ}_i \wedge \neg \text{HBUSREQ}_j \wedge \text{HMASTER} \neq i \wedge \text{HMASTER} \neq j)(\text{HBUSREQ}_i \wedge \text{HMASTER} \neq i)^*(\text{HMASTER} = j)$.

One may suggest to use the $GR(1)^*$ $GEF(reg)$ operator in order to express use case D . However, as we show next, this suggestion may not reflect the engineer's intention. The existential guarantee $GEF(reg)$ expresses the requirement that *from every reachable state, the described scenario can be eventually satisfied*. But, the engineer's intention was to express the requirement that *every prefix of a play can be completed into a play that, eventually, satisfies the scenario*.

Note that these two requirements are indeed different. For illustration, consider a prefix of a play, $\pi(0), \dots, \pi(i)$, such that $\pi(i-1) \models (\text{HBUSREQ}_i \wedge \neg \text{HBUSREQ}_j \wedge \text{HMASTER} \neq i \wedge \text{HMASTER} \neq j)$, and $\pi(i) \models (\text{HBUSREQ}_i \wedge \text{HMASTER} \neq i)$. Extending this prefix with a continuation of the form $(\text{HBUSREQ}_i \wedge \text{HMASTER} \neq i)^t(\text{HMASTER} = j)$ meets the engineer's intention, since the concatenation of the play's prefix with such a continuation satisfies reg . However, this intention is not captured by $GEF(reg)$. $GEF(reg)$ expresses the requirement that there exists a way to satisfy reg , starting from a future state. In contrast, the operator we shall shortly define, $GE^+(reg)$, means that the states traversed so far can be concatenated to a possible future behavior that satisfies reg (and hence we choose the notation '+' to denote this concatenation).

We now extend $GR(1)^*$ with a new operator, $GE^+(reg)$. Specifically, in Def. 7.1 below we extend Def. 4.2 to support formulas of the form

$$A\left(\bigwedge_{j=1}^m GF(a_j) \rightarrow \left(\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k) \wedge \bigwedge_{t=1}^r GE^+(\widehat{reg_t})\right)\right),$$

where $a_1, \dots, a_m, g_1, \dots, g_n$ are assertions over $\mathcal{V}$, and $reg_1, \dots, reg_\ell, \widehat{reg_1}, \dots, \widehat{reg_r}$ are regular expressions over $\mathcal{V}$.

Definition 7.1 Consider a game structure $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, and a formula $\psi = A\left(\bigwedge_{j=1}^m GF(a_j) \rightarrow \left(\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k) \wedge \bigwedge_{t=1}^r GE^+(\widehat{reg_t})\right)\right)$ over $\mathcal{X} \cup \mathcal{Y}$. Let f^s be a strategy for the system, and $\pi = \pi(0), \pi(1), \dots$ a play in GS , consistent with f^s . The pair (f^s, π) wins for the system w.r.t. ψ in GS if one of the following holds:

1. π is finite and it reaches a deadlock for the environment.
2. π is infinite and $\pi \models FG(\neg a_j)$ for some justice assumption a_j .
3. π is infinite, and the following hold:

$$\bigwedge_{i=1}^n GF(g_i): \pi \models \bigwedge_{i=1}^n GF(g_i).$$

$\bigwedge_{k=1}^\ell GEF(reg_k)$: For every $i \geq 0$, and an existential guarantee $GEF(reg_k)$, the i -th prefix of $\pi, \pi(0), \dots, \pi(i)$, can be extended to a play $\tilde{\pi}$, consistent with f^s , such that there exist $i \leq i_0 \leq j$, with $\tilde{\pi}(i_0) \cdots \tilde{\pi}(j) \in L(reg_k)$.

$\bigwedge_{t=1}^r GE^+(\widehat{reg}_t)$: For every $i \geq 0$, and an existential guarantee $GE^+(\widehat{reg}_t)$, the i -th prefix of $\pi, \pi(0), \dots, \pi(i)$, can be extended to a play $\tilde{\pi}$, consistent with f^s , such that there exists $j \geq i$, with $\tilde{\pi}(0) \cdots \tilde{\pi}(j) \in L(\widehat{reg}_t)$.

We support existential guarantees of the form $GE^+(\widehat{reg})$ via a reduction to GR(1)* formulas, as were defined in Sect. 4. Specifically, we show that $GE^+(\widehat{reg})$ can be expressed by the GEF operator. Our reduction involves the addition of a new (non-Boolean) auxiliary variable, along with a corresponding modification of the transition relation ρ^s . We modify ρ^s to update the value of the newly added variable in each step, so that a certain assertion $assrt$ holds exactly when the play's prefix, traversed so far, satisfies the regular expression $\widehat{reg}$. This allows us to replace $GE^+(\widehat{reg})$ with the existential guarantee $GEF(assrt)$.

The reduction is formally defined as follows. Take a DFA, $\mathcal{A}_{\widehat{reg}} = (Q, 2^{\mathcal{X} \cup \mathcal{Y}}, \delta, q_0, \mathcal{F})$ with $L(\mathcal{A}_{\widehat{reg}}) = L(\widehat{reg})$. We add a new variable y , controlled by the system, that stores the current state of the automaton,² and we replace θ^s and ρ^s with $\tilde{\theta}^s$ and $\tilde{\rho}^s$, respectively. Let $vals(y) = \{0, \dots, |Q| - 1\}$ be the set of values that y may store. For $s \in 2^{\mathcal{X} \cup \mathcal{Y}}$ and $u \in vals(y)$, $(s, u) \models \tilde{\theta}^s$ if $s \models \theta^s$, and $\delta(q_0, s) = q_u$. In words, $(s, u) \models \tilde{\theta}^s$ if $s \models \theta^s$, and by reading s from the initial state q_0 , the automaton traverses to state q_u .

Now, for every transition $\delta(q_u, t) = q_v$, let $Trans_{(u,t)}$ be the assertion $Trans_{(u,t)} := (y = u) \wedge t' \wedge (y' = v)$, and let $\tilde{\rho}^s := \rho^s \wedge (\bigvee_{(u,t) \in vals(y) \times 2^{\mathcal{X} \cup \mathcal{Y}}} Trans_{(u,t)})$. In words, $((s, u), (t', v')) \models \tilde{\rho}^s$ if $(s, t') \models \rho^s$, and when the automaton reads t from state q_u , it traverses to q_v .

Clearly, for a play-prefix $\pi(0), \dots, \pi(i)$, we have $\pi(0) \cdots \pi(i) \in L(\widehat{reg})$ iff $\pi(i) \models y \in \mathcal{F}$. Hence, in the updated game structure, the existential guarantee $GE^+(\widehat{reg})$ is replaced with the existential guarantee $GEF(y \in \mathcal{F})$.

Remark 7.2 Note that unlike in the case of an existential guarantee $GEF(reg)$, a DFA that recognizes reg is crucial for the construction, and it cannot be replaced with an NFA. The reason is that with an NFA, the system can mistakenly track an “incorrect” path, and thus incorrectly conclude that reg has not been satisfied. In particular, it is possible that at runtime, the system will not be able to choose the “correct path”, because it depends on the future behavior of the environment. We therefore follow [MR15a] here, and use DFAs to track the satisfaction of the regular expressions.

7.2. A special case: past LTL

An important special case of the $GE^+(reg)$ extension is the support for past LTL formulas. The authors of [BJP⁺12] noted that atomic past LTL formulas can be tested by Boolean auxiliary variables, and described how the GR(1) scheme can be extended to support past LTL assumptions and guarantees.

A past LTL formula is generated by the grammar

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid Y\varphi \mid O\varphi \mid H\varphi \mid \varphi S\varphi$$

where p is an assertion, and Y, O, H, S represent the temporal operators *previously*, *once*, *historically*, and *since*, respectively. For $\pi(0), \dots, \pi(i) \in (2^{\mathcal{X} \cup \mathcal{Y}})^+$, the semantics of a past LTL formula is defined as follows:

- $\pi(0), \dots, \pi(i) \models p$ if $\pi(i) \models p$;
- $\pi(0), \dots, \pi(i) \models Y\varphi$ if $i > 0$, and $\pi(0), \dots, \pi(i-1) \models \varphi$;
- $\pi(0), \dots, \pi(i) \models O\varphi$ if $\exists j \leq i(\pi(0), \dots, \pi(j) \models \varphi)$;
- $\pi(0), \dots, \pi(i) \models H\varphi$ if $\forall j \leq i(\pi(0), \dots, \pi(j) \models \varphi)$;
- $\pi(0), \dots, \pi(i) \models \varphi_1 S \varphi_2$ if $\exists j \leq i(\pi(0), \dots, \pi(j) \models \varphi_2 \wedge \forall k \in \{j+1, \dots, i\}(\pi(0), \dots, \pi(k) \models \varphi_1))$.

In [KPR98] it was shown how to construct testers for the operators Y and S . We repeat that construction and extend it to O and H as well. Consider a game structure, $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, and assertions, a and b . For each formula $Y(a)$, $O(a)$, $H(a)$, and aSb , we show how to replace it with an auxiliary variable y that indicates when the original formula holds. This is done by modifying the game structure as follows:

²Hence, actually, a value of y represents an assignment to $\log |Q|$ new Boolean variables.

$Y(a)$. θ^s is replaced with $\theta^s \wedge \neg y$. ρ^s is replaced with $\rho^s \wedge (y' \leftrightarrow a)$.
 $O(a)$. θ^s is replaced with $\theta^s \wedge (y \leftrightarrow a)$. ρ^s is replaced with $\rho^s \wedge (y' \leftrightarrow (y \vee a'))$.
 $H(a)$. θ^s is replaced with $\theta^s \wedge (y \leftrightarrow a)$. ρ^s is replaced with $\rho^s \wedge (y' \leftrightarrow (y \wedge a'))$.
 aSb . θ^s is replaced with $\theta^s \wedge (y \leftrightarrow b)$. ρ^s is replaced with $\rho^s \wedge (y' \leftrightarrow (b' \vee (y \wedge a')))$.

By this construction, nested formulas can be unfolded to be tested by auxiliary variables. As an example, we consider again our regular expression, $reg = \text{true}^*(\text{HBUSREQ}_i \wedge \neg \text{HBUSREQ}_j \wedge \text{HMASTER} \neq i \wedge \text{HMASTER} \neq j)(\text{HBUSREQ}_i \wedge \text{HMASTER} \neq i)^*(\text{HMASTER} = j)$. This regular expression is equivalent to the past LTL formula:

$$\begin{aligned}
 &(\text{HMASTER} = j) \wedge Y((\text{HBUSREQ}_i \wedge \text{HMASTER} \neq i)S \\
 &(\text{HBUSREQ}_i \wedge \neg \text{HBUSREQ}_j \wedge \text{HMASTER} \neq i \wedge \text{HMASTER} \neq j)).
 \end{aligned}$$

The tester for $(\text{HBUSREQ}_i \wedge \text{HMASTER} \neq i)S(\text{HBUSREQ}_i \wedge \neg \text{HBUSREQ}_j \wedge \text{HMASTER} \neq i \wedge \text{HMASTER} \neq j)$ is y_1 , and $Y(y_1)$ is tested by y_2 . Therefore:

1. We replace θ^s with $\theta^s \wedge (y_1 \leftrightarrow (\text{HBUSREQ}_i \wedge \neg \text{HBUSREQ}_j \wedge \text{HMASTER} \neq i \wedge \text{HMASTER} \neq j)) \wedge (\neg y_2)$.
2. We replace ρ^s with $\rho^s \wedge (y'_1 \leftrightarrow ((\text{HBUSREQ}'_i \wedge \neg \text{HBUSREQ}'_j \wedge \text{HMASTER}' \neq i \wedge \text{HMASTER}' \neq j) \vee (y_1 \wedge (\text{HBUSREQ}'_i \wedge \text{HMASTER}' \neq i)))) \wedge (y'_2 \leftrightarrow y_1)$.

This way, the engineer integrates the existential guarantee $GE^+(reg)$ into the specification, with the cost of adding two Boolean variables, by replacing $GE^+(reg)$ with the existential guarantee $GEF(\text{HMASTER} = j \wedge y_2)$. In general, this construction enables supporting existential guarantees of the form $GE^+(\varphi)$ where φ is a past LTL formula, and each past LTL sub-formula of φ adds a new single variable to the game structure.

7.3. Supporting additional forms of existential guarantees

$GEF(reg)$ means that from every state we can satisfy reg , starting from a future state, while $GE^+(reg)$ means that from every state, we are able to satisfy reg , starting from the first state of the play.

Another natural form of an existential guarantee that the engineer may wish to express is to require that reg can be satisfied by concatenating a suffix of the sequence of states traversed so far, to some future behavior. That is, the engineer may require that for any play π and $i \geq 0$, there exists a play $\tilde{\pi}$ such that $(\pi(0), \dots, \pi(i)) = (\tilde{\pi}(0), \dots, \tilde{\pi}(i))$, and for which there exist i_0, j , such that $i_0 \leq j$, $i \leq j$, and $\tilde{\pi}(i_0) \dots \tilde{\pi}(j) \in L(reg)$. Note that this requirement differs from $GEF(reg)$ since we include the possibility $i_0 < i$. Thanks to the $GE^+(reg)$ defined above, $\text{GR}(1)^*$ supports such form of an existential guarantee by writing $GE^+(\text{true}^*reg)$.

Another possible requirement the engineer may wish to express is that every play-prefix can be extended to a play that includes a subword that satisfies reg . That is, to require that for every play π and $i \geq 0$, $\pi(0), \dots, \pi(i)$ can be extended to a play $\tilde{\pi}$, for which there exist i_0, j , such that $i_0 \leq j$, and $\tilde{\pi}(i_0) \dots \tilde{\pi}(j) \in L(reg)$. This requirement differs from previous forms of existential guarantees by including the possibility $j < i$. Again, thanks to the $GE^+(reg)$ defined above, $\text{GR}(1)^*$ supports such form of an existential guarantee by writing $GE^+(\text{true}^*reg \text{ true}^*)$.

8. The well-separated case

In some cases, the system can win a $\text{GR}(1)^*$ game from a certain state by forcing a violation of the assumptions. Specifications that allow this possibility, termed *non-well-separated* [KP10], are considered to be problematic. Maoz and Ringert [MR16] showed a technique to detect non-well-separation in $\text{GR}(1)$ specifications. As non-well-separation is a property of the assumptions in the specification, the technique works for $\text{GR}(1)^*$ as well.

In this section we show that when the specification is well-separated, the $\text{GR}(1)^*$ controller achieves more than required. First, the existential guarantees hold regardless of the satisfaction or dissatisfaction of the environment's justice assumptions. Second, any existential guarantee $GEF(reg)$ is satisfied iff a stronger existential

guarantee $GEF((reg \text{ true}^*)^\omega)$ is satisfied. Finally, we show that when the specification is well-separated, GR(1)* can support existential guarantees with deterministic Büchi properties, which enable the specification of patterns [MR15a].

8.1. Achieving more than required in a well-separated environment

Inspired by the definition of well-separation for GR(1) games [KP10], we define:

Definition 8.1 (Well-separated environment) Let $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$ be a game structure and let $a_1, \dots, a_m$ be assertions over $\mathcal{X} \cup \mathcal{Y}$. The pair $(GS, \{a_1, \dots, a_m\})$ has a well-separated environment if from every $s \in 2^{\mathcal{X} \cup \mathcal{Y}}$, the environment has a strategy to force satisfaction of the formula $\bigwedge_{j=1}^m GF(a_j)$.

Consider a GR(1)* formula $A(\bigwedge_{j=1}^m GF(a_j) \rightarrow (\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k)))$. We argue that in case of a well-separated environment, a winning strategy achieves more than required and the winning condition $A((\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)) \wedge \bigwedge_{k=1}^\ell GEF((reg_k \text{ true}^*)^\omega))$ is satisfied. This formula differs from the GR(1)* formula in two ways: (1) the modified existential guarantees appear after the conjunction operator $\wedge$, and not as an implication of the $\rightarrow$ operator; (2) each existential guarantee $GEF(reg)$ is replaced with the stronger $GEF((reg \text{ true}^*)^\omega)$. This means that (1) any of the existential guarantees should hold from any reachable state, and not just along plays that satisfy the justice assumptions; (2) the controller and the environment can satisfy, cooperatively, the stronger existential guarantee $GEF((reg \text{ true}^*)^\omega)$.

Definition 8.2 (Stronger winning condition) Let $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$ be a game structure and $\psi = A((\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)) \wedge \bigwedge_{k=1}^\ell GEF((reg_k \text{ true}^*)^\omega))$ where $a_1, \dots, a_m, g_1, \dots, g_n$ are assertions over $\mathcal{X} \cup \mathcal{Y}$, and $reg_1, \dots, reg_\ell$ are regular expressions over $\mathcal{X} \cup \mathcal{Y}$. A strategy for the system f^s wins from a state s w.r.t. ψ in GS , if for any play π from s , consistent with f^s , the following hold:

1. π is infinite.
2. If $\pi \models \bigwedge_{j=1}^m GF(a_j)$, then $\pi \models \bigwedge_{i=1}^n GF(g_i)$.
3. For every $k \in \{1, \dots, \ell\}$, and $i \geq 0$ there exists a play $\tilde{\pi}$, consistent with the strategy, and $j \geq i$, such that $(\tilde{\pi}(0), \dots, \tilde{\pi}(i)) = (\pi(0), \dots, \pi(i))$, and $\tilde{\pi}(j)\tilde{\pi}(j+1) \dots \in L((reg_k \text{ true}^*)^\omega)$.

Lemma 8.3 Let $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$ be a game structure, $\phi = A(\bigwedge_{j=1}^m GF(a_j) \rightarrow (\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k)))$ a GR(1)* formula, and f^s a strategy for the system player. If the pair $(GS, \{a_1, \dots, a_m\})$ has a well-separated environment, then f^s is a winning strategy w.r.t. ϕ in GS iff it is a winning strategy w.r.t. $\psi = A((\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)) \wedge \bigwedge_{k=1}^\ell GEF((reg_k \text{ true}^*)^\omega))$ in GS .

Therefore, in case of a well-separated environment, the controller we construct is robust against assumption violations: it ensures that the regular expressions of the existential guarantees can be satisfied infinitely many times from any reachable state, in all plays, and not only in plays that satisfy the justice assumptions. In particular, this means that the system must avoid any kind of deadlock.

To illustrate this robustness, consider our running example from Sect. 2, with additional assumptions that make it non-well-separated. First, consider the following safety assumption:

$$G(\bigvee_{i=1}^n \text{HBUSLOCK}_i \rightarrow \bigwedge_{i=1}^n (\text{HBUSREQ}_i = 0 \rightarrow X(\text{HBUSREQ}_i = 0))),$$

where n is the number of masters. This means that when some master asks for a locked-access, no master is allowed to raise a new request. Second, consider the following justice assumptions:

$$GF(\text{HBUSREQ}_i),$$

for $i \in \{1, \dots, n\}$, which assumes that at any point in time, each master will eventually ask to access the bus.

Now, assume that at some point, master 1 uses the bus, and master 2 asks for a locked-access. In this case, a synthesized controller may decide to act as follows: it waits until master 1 finishes to transfer its data, and then it grants no requests for the rest of the play. By doing that, this controller forces violation of the justice assumptions: since master 2 asks for a locked-access, master 1 is not allowed to raise a request, due to the added safety assumption. As a result, the added justice assumption $GF(\text{HBUSREQ}_1)$ is violated. Note that the behavior of this synthesized controller is undesirable not only because it does not satisfy its justice guarantees, but also because, from some point on, its existential guarantees cannot be realized anymore.

Using the technique proposed in [MR16], the engineer can identify that the added assumptions caused non-well-separation. She can decide to remove the problematic justice assumptions, and the environment is again well-separated. This time, the controller cannot force violation of the assumptions from any state and thus it will be obligated to satisfy each existential guarantee $GEF(\text{reg}_k)$ from any reachable state. Therefore, the engineer now knows that the synthesized controller will satisfy the stronger existential guarantees $GEF((\text{reg}_k \text{ true}^*)^\omega)$.

8.2. Deterministic Büchi properties and patterns as existential guarantees

Maoz and Ringert [MR15a] showed how to extend the GR(1) fragment to support deterministic Büchi properties as justice guarantees. That is, to support ω -regular properties that can be accepted by a DBW. This includes 52 out of the 55 patterns suggested by Dwyer et al. [DAC99], e.g., the well-known and useful response pattern $G(\text{req} \rightarrow F(\text{ack}))$, accepted by a 2-state DBW. We will now show that the construction of [MR15a] (outlined in Sect. 7) and the result of Sect. 8.1 enable, in case the specification is well-separated, to integrate such properties as existential guarantees, with the semantics of the GE^+ operator from Sect. 7. The cost of this extension is only logarithmic in the number of states of the DBW that recognizes the property: we add $\log |Q|$ variables to the game structure. Since according to [MR15a], any of these patterns is accepted by a DBW with at most 8-states, any of the 52 patterns mentioned above induces a cost of adding, at most, three Boolean variables to the game structure.

For illustration, consider our example from Sect. 2. Suppose that the engineer wants to add to the specification the use case in which master 1 accesses the bus exactly once. Note that this property is accepted by a DBW with 3 states.

Use case E. Master 1 accesses the bus exactly once:

$$GE^+(\neg(\text{HMASTER} = 1)^*(\text{HMASTER} = 1)^+(\neg(\text{HMASTER} = 1))^\omega).$$

Also, the engineer integrates properties that form special cases of the common patterns: (1) a request leads to an acknowledgment - $G(\text{req} \rightarrow F(\text{ack}))$, and (2) p occurs between q and r - $G((q \wedge \neg r) \rightarrow (\neg r W(p \wedge \neg r)))$, where the operator ' $\varphi W \psi$ ' (named *weak until*) is an abbreviation for $(\varphi U \psi) \vee (G\varphi)$.

Use case F. Whenever a request by master 1 is granted, a request by master n will eventually be granted afterwards.

$$GE^+(G(\text{HMASTER} = 1 \rightarrow F(\text{HMASTER} = n))).$$

Use case G_i. Between an access to the bus by master 1, and an access to the bus by master 2, master i accesses the bus:

$$GE^+(G((\text{HMASTER} = 1 \wedge \neg(\text{HMASTER} = 2)) \rightarrow (\neg(\text{HMASTER} = 2) W((\text{HMASTER} = i) \wedge \neg(\text{HMASTER} = 2))))).$$

More generally, we wish to support formulas of the form:

$$A\left(\left(\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)\right) \wedge \bigwedge_{k=1}^{\ell} GEF((\text{reg}_k \text{ true}^*)^\omega) \wedge \bigwedge_{t=1}^r GE^+(\psi_t)\right),$$

where each ψ_t is a deterministic Büchi property. $GE^+(\psi_t)$ means that for each play π and $i \geq 0$, the prefix of the play $\pi(0) \dots \pi(i)$ can be extended into a play π' , consistent with the system's strategy, such that $\pi' \models \psi_t$ (see Sect. 7 and Def. 7.1 for a discussion about the GE^+ operator).

To support this extension, we use the technique outlined in Sect. 7. Consider a game structure $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, and assume that the pair $(GS, \{a_1, \dots, a_m\})$ has a well-separated environment. For each $t \in \{1, \dots, r\}$, let $\mathcal{A}_t = (Q^t, 2^{\mathcal{X} \cup \mathcal{Y}}, \delta^t, q_0^t, \mathcal{F}^t)$ be a DBW that recognizes the property ψ_t . We extend the game structure into $\tilde{GS} = (\mathcal{X}, \mathcal{Y} \cup \{y_1, \dots, y_r\}, \theta^e, \tilde{\theta}^s, \rho^e, \tilde{\rho}^s)$ as in Sect. 7. Thus, plays in GS and in $\tilde{GS}$ correspond to each other:

- If $\tilde{\pi}$ is a play in $\tilde{GS}$, then the play π , defined by $\pi(i) = \tilde{\pi}(i) \upharpoonright_{\mathcal{X} \cup \mathcal{Y}}$, is a play in GS .
- If π is a play in GS , then there is a unique play $\tilde{\pi}$ in $\tilde{GS}$ such that $\tilde{\pi}(i) \upharpoonright_{\mathcal{X} \cup \mathcal{Y}} = \pi(i)$. Further, for each $t \in \{1, \dots, r\}$ the value of y_t in $\tilde{\pi}(j)$ is v iff $q_0^t \xrightarrow{\pi(0) \dots \pi(j)} q_v^t$.

Therefore, a strategy f^s for the system in $\tilde{GS}$ also constitutes a strategy for the system in the game structure GS . For $f^s(s_0, \dots, s_k, s_x) = s_y$, we naturally define $f^s(s_0 \upharpoonright_{\mathcal{X} \cup \mathcal{Y}}, \dots, s_k \upharpoonright_{\mathcal{X} \cup \mathcal{Y}}, s_x) = s_y \upharpoonright_{\mathcal{Y}}$. The following lemma is implied by the construction elaborated in Sect. 7 and by Lem. 8.3.

Lemma 8.4 Let f^s be a strategy for the system in $\tilde{GS}$. If the pair $(GS, \{a_1, \dots, a_m\})$ has a well-separated environment, then f^s is a winning strategy w.r.t. $\phi = A((\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)) \wedge \bigwedge_{k=1}^\ell GEF((reg_k \text{true}^*)^\omega) \wedge \bigwedge_{t=1}^r GE^+(\psi_t))$ in GS iff f^s is a winning strategy w.r.t. $\psi = A(\bigwedge_{j=1}^m GF(a_j) \rightarrow (\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k) \wedge \bigwedge_{t=1}^r GEF(y_t \in \mathcal{F}^t)))$ in $\tilde{GS}$.

Roughly, the lemma is proved using the following arguments. First, relying on Lem. 8.3, the well-separation property implies that a winning strategy for the system in $\tilde{GS}$ satisfies the stronger existential guarantees $GEF(((y_t \in \mathcal{F}^t)_{\text{true}^*})^\omega)$. Moreover, as explained in Sect. 7, the assertion $y_t \in \mathcal{F}^t$ holds in $\tilde{GS}$ iff the play-prefix traversed leads the automaton to an $\mathcal{F}^t$ -state. Hence, $GEF(((y_t \in \mathcal{F}^t)_{\text{true}^*})^\omega)$ holds iff $GE^+(\psi_t)$ holds. A detailed proof is given in Appx. A.

9. Unrealizability of GR(1)* games

It is well known that ω -regular games (and thus LTL games and in particular GR(1) games) are determined [GTW02]; the winning regions of the system and the environment form a partition of the state space. Previous work on GR(1) has relied on its determinacy in the context of dealing with unrealizability. Specifically, to analyze and understand reasons for unrealizability, researchers suggested to construct a *counter-strategy*—a winning strategy for the environment from the complement of the system's winning region [KHB13, KMR17].

GR(1)* is not a fragment of LTL. Do unrealizable GR(1)* games always admit counter-strategies? In this section we prove a crucial result about the difficulty of tackling unrealizability of GR(1)* specifications. We show the existence of GR(1)* games for which both players have no winning strategy from certain states, i.e., GR(1)* games for which the winning regions of the system and the environment do not form a partition of the state space.

In Sect. 5 we characterized the exact set of states from which the system has a winning strategy in a GR(1)* game via a μ -calculus formula (Eq. 4), and in Sect. 6 we provided a controller construction from the system's winning region. However, as stems from what we prove below, the negation of the formula in Eq. 4 does not compute the set of states from which the environment has a winning strategy. In other words, in some GR(1)* games, some states belong neither to the winning region of the system player nor to the winning region of the environment player. From these states, for any strategy for the system f^s , the environment has a strategy f^e that yields a play π such that (f^s, π) does not win for the system, and vice-versa, for any strategy for the environment f^e , the system has a strategy f^s that yields a play π such that (f^s, π) wins for the system.

To clarify, one should not confuse the result we outlined above with the soundness and completeness of the solution we presented in Sect. 5. Given a GR(1)* specification, we are always able to compute the states from which the system can win: the formula in Eq. 4 fully computes the winning region for the system. The possible absence of a counter-strategy means that, in some cases, some states in the complement of this winning region are not winning for the environment.

Before we provide the relevant definitions and our result, we note a fundamental difference between $\text{GR}(1)^*$ games and general CTL^* games. Previous works that study CTL^* games define a winning strategy as one whose computation tree satisfies the winning condition, given as a CTL^* formula [BSK17, KV99, KV00]. Hence, the system cannot win from a state s w.r.t. a CTL^* winning condition φ , if for any strategy f^s for the system from s , the computation tree induced by f^s , T_{f^s} , satisfies: $T_{f^s} \not\models \varphi$. This definition does not take into account the choices made by the environment during a specific play. Therefore, in this context, the notion of a counter-strategy is meaningless. In contrast, Def. 4.2 distinguishes between winning and losing plays that are consistent with the same strategy. This allows us to define a counter-strategy for $\text{GR}(1)^*$ games in a standard manner, i.e., as one that wins in every play, regardless of the strategy chosen by the system.

Definition 9.1 (GR(1)* counter-strategy) Let $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$ be a game structure, ψ a $\text{GR}(1)^*$ formula, and f^e a strategy for the environment. f^e wins for the environment from state s , if for every strategy for the system f^s from s , for the unique play π , consistent with f^e and f^s , it holds that (f^s, π) does not win for the system w.r.t. ψ .

We show through the next example that unlike $\text{GR}(1)$ games, a $\text{GR}(1)^*$ game may admit states that do not belong to the winning region of any of the players.

Example 9.2 Let $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$ be a game structure, and $\psi = A(GF(\text{true}) \rightarrow (GF(g) \wedge GEF(q)))$ a $\text{GR}(1)^*$ formula, where

- $\mathcal{X} = \{x\}$.
- $\mathcal{Y} = \{g, q\}$.
- $\theta^e = \theta^s = \rho^e = \text{true}$.
- $\rho^s = (g' \leftrightarrow \neg q') \wedge (q \rightarrow q')$.

In Ex. 9.2, in each step the system assigns true to exactly one of the variables among $\{g, q\}$. Further, once the system assigns true to q , it must assign true to q in all ensuing states.

Lemma 9.3 In Ex. 9.2, for every $s \in 2^{\mathcal{X} \cup \mathcal{Y}}$, the system has no winning strategy from s .

Clearly, the environment has a winning strategy from the set $\{s \in 2^{\mathcal{X} \cup \mathcal{Y}} : s \models q\}$. This holds since if $s \models q$, then any play π from s that is consistent with $\rho^e \wedge \rho^s$, satisfies $\pi \models \neg(GF(g))$. Hence, in fact, any strategy for the environment is a winning strategy from $\{s \in 2^{\mathcal{X} \cup \mathcal{Y}} : s \models q\}$. It is left to prove that the $\text{GR}(1)^*$ game of Ex. 9.2 admits states from which the environment does not win. Specifically, we show that the environment has no winning strategy from every state s such that $s \models \neg q$.

Lemma 9.4 In Ex. 9.2, take $s \in 2^{\mathcal{X} \cup \mathcal{Y}}$ and a strategy for the environment f^e . If $s \models \neg q$, then the system has a strategy f^s , such that for the unique play π that is consistent with f^s and f^e , it holds that (f^s, π) wins for the system.

Lemmas 9.3 and 9.4 imply the main result of this section, which may be viewed as a form of indeterminacy.

Theorem 9.5 There exist a $\text{GR}(1)^*$ game and a state s , such that each of the two players, the system and the environment, has no winning strategy from s .

As a consequence, when dealing with $\text{GR}(1)^*$ games, existing techniques related to unrealizability, e.g., the examination of a counter-strategy, cannot be used directly. Thus, Thm. 9.5 induces future research directions. First, it is important to compute the set of states from which the environment has a winning strategy, and the set of states from which neither players has a winning strategy. Second, as the construction of a counter-strategy from the environment's winning region may help in understanding unrealizability, unrealizable $\text{GR}(1)^*$ specifications for which the environment's winning region is empty may pose a special challenge that requires further investigation.

10. Implementation and evaluation

We have implemented realizability checking for $\text{GR}(1)^*$ from Sect. 5, and integrated it in the Spectra language and synthesis environment [MR21, Spe], which already includes a $\text{GR}(1)$ synthesizer and implementations of several additional analyses. Our implementation uses BDDs [Bry86] via the CUDD 3.0 [Som15] package. The implementation includes the heuristics for $\text{GR}(1)$ realizability checking detailed in [FMR20].

10.1. Validation

We validated the correctness of our implementation as follows. First, to validate the correctness of the translation of regular expressions to NFAs, we implemented two separate translation schemes. The first is a pure BDD-based approach that constructs a symbolic Epsilon-NFA and then eliminates all of the epsilon transitions to obtain a symbolic NFA [VdHT10]. The second is an approach that uses the well-known Brics package.³ While the second approach suffers from an inferior running time performance (as the translation to Brics regular expressions entails an exponential blowup of the alphabet), we used it as an oracle for validating our first translation scheme, which we used in our evaluation. Specifically, we randomly generated thousands of regular expressions and checked that the NFAs constructed by both approaches accept the same languages.

Second, to validate the correctness of the GR(1)* realizability checking implementation we tested it on many small synthetic specifications for which we were able to check that the realizability results reported by the implementation matched the ones we expected.

We make the implementation and all the specifications used in our experiments below available for inspection and reproduction.⁴

10.2. Research questions

To evaluate the efficiency of our proposed technique of supporting existential guarantees and its implementation, we consider the following research questions:

RQ1. What is the cost, in terms of realizability checking running time, of supporting existential guarantees as part of GR(1) specifications? Does the technique scale well?

RQ2. Are some existential guarantees more costly than others?

10.3. Corpus and experiment setup

As the corpus for our experiments, we took the family of AMBA specifications used in our running example, where the number of masters ranges from 2 to 5. We denote by $AMBA_n$ the AMBA specification with n masters. Note that in addition to the variables listed in Sect. 2, each specification includes auxiliary variables that are used to express pattern properties, i.e., requirements that are neither safety nor justice properties (specifically, requirements A1, G2, G3, and G10 in [BGJ⁺07]).

Table 1 presents the state space, number of justice assumptions, and number of justice guarantees, for AMBA2 to AMBA5. It further shows the realizability checking running time for each of the four specifications. We used these as a baseline for our evaluation.

On top of the baseline specifications AMBA2 to AMBA5, we created new specifications with existential guarantees. Considering the use cases A, B, and C from Sect. 2, we added to the baseline specifications existential guarantees of the form $GEF(reg)$, where reg is a regular expression of type A, B, or C. We divide all specifications into sets as follows.

SET1. For each $n \in \{2, 3, 4, 5\}$, a regular expression type $X \in \{A, B, C\}$, and $k \in \{1, \dots, n\}$, we created a configuration by adding to $AMBA_n$ k different existential guarantees of the form $GEF(reg)$, where reg is of type X . Hence, SET1 includes 42 specifications.

SET2. For each $n \in \{2, 3, 4, 5\}$ and $k \in \{1, \dots, n\}$, we created a configuration by adding to $AMBA_n$ $3k$ different existential guarantees of the form $GEF(reg)$: k regular expressions of type A, k of type B, and k of type C. Hence, SET2 includes 14 specifications.

³<https://www.brics.dk/automaton/>

⁴<http://smlab.cs.tau.ac.il/syntech/existential/>

Table 1. Statistics on the baseline configurations and their realizability checking running times in Spectra

	AMBA2	AMBA3	AMBA4	AMBA5
State space	2^{22}	2^{28}	2^{34}	2^{40}
# Justice assumptions	2	2	2	2
# Justice guarantees	5	7	9	11
Running time (ms)	437	6258.5	63542	78586

Then, we wanted to create larger specifications with more existential guarantees. Note that we cannot add to AMBA_n more than n existential guarantees of type B, but we can add up to $n^2 - n$ existential guarantees of type A and C, as this is the exact number of ordered pairs of different masters. We therefore created another set as follows.

SET3. For each $n \in \{2, 3, 4, 5\}$ and $k \in \{n + 1, \dots, n^2 - n\}$, we created a configuration by adding to AMBA_n $n + 2k$ existential guarantees: n different existential guarantees of type B, k different existential guarantees of type A, and k different existential guarantees of type C. Hence, SET3 includes 26 specifications.

Overall, excluding the baseline specifications, we considered 82 different configurations with up to 45 existential guarantees. The lengths of the formulas representing the use cases A, B, and C are 9, 3, and 10 respectively.⁵ Hence, we added existential guarantees of total length up to 395 symbols, i.e., the total length of the added existential guarantees' regular expressions.

For each configuration, including the baseline specifications, we repeated realizability checking 10 times.⁶ Each time, we randomly chose master indexes as values, to turn a parameterized configuration into a concrete specification. We measured realizability checking running times. We report median values of the 10 executions. We performed all the experiments on a rather ordinary desktop computer with an Intel Xeon W-2133 processor, 32 GB RAM, running Windows 10.

10.4. Results

Table 1 presents the median running times for the baseline specifications AMBA2 to AMBA5.

Tables 2 and 3 present the ratios between the running time for the different configurations and the baseline. Table 2 refers to configurations of SET1, i.e., extensions of the baseline specifications with existential guarantees of a single type. Table 3 presents the results of all experiments in which the specifications are extended by several existential guarantees of each type. Table 3 is divided into two parts, separated by a horizontal line. The upper part refers to experiments with $k = 1, \dots, n$ existential guarantees of each type (SET2), and the lower part refers to experiments in which we added n existential guarantees of type B, and $k > n$ existential guarantees of types A and C (SET3).

For illustration, as an example, in Table 3, the value of the cell on the column titled AMBA4 and the row titled 10, n , 10 is 135%. This means that when we augmented AMBA4 with 10 existential guarantees of type A, $n = 4$ existential guarantees of type B, and 10 existential guarantees of type C, the median of 10 runs of realizability checking took 35% longer than the baseline, i.e., about 88 seconds.

Finally, Fig. 2 presents the running times ratios as a function of the number of existential guarantees added to each baseline specification, based on the results with the multi-type configurations (SET2 and SET3), as presented in Table 3.

To answer RQ1, based on the results of our experiments, we observe that adding existential guarantees to a $\text{GR}(1)$ specification, as expected, has a cost. Yet, the results reveal that $\text{GR}(1)^*$ realizability checking is feasible. In 70 out of the 82 configurations, the running time was less than twice the running time of the corresponding baseline configuration. Among the remaining 12 configurations, only for the “heaviest” one, the running time was more than three times than the baseline configuration, specifically 3.29. Note that this specification includes 45 existential guarantees and only 13 justice constraints (assumptions and guarantees) from the corresponding baseline specification AMBA5.

⁵Assertions are counted as a single symbol.

⁶Even though the algorithm is deterministic, we performed multiple runs since the default automatic variable reordering of CUDD adds variance to running times.

Table 2. Ratios between the running times of configurations with single type additional existential guarantees (SET1) and the running times of the baseline configurations

Type	Number of added existential guarantees				
	1	2	3	4	5
AMBA2					
A	104%	119%			
B	110%	95%			
C	105%	115%			
AMBA3					
A	121%	135%	127%		
B	91%	104%	97%		
C	99%	124%	124%		
AMBA4					
A	84%	99%	90%	87%	
B	83%	84%	91%	106%	
C	93%	93%	95%	96%	
AMBA5					
A	128%	150%	129%	126%	102%
B	148%	109%	121%	117%	141%
C	108%	109%	129%	128%	106%

Table 3. Ratios between the running times of configurations with multi-type additional existential guarantees (SET2 and SET3), and the running times of the baseline configurations

# Existential guarantees	AMBA2	AMBA3	AMBA4	AMBA5
1, 1, 1	129%	115%	102%	127%
2, 2, 2	163%	137%	95%	116%
3, 3, 3		156%	102%	165%
4, 4, 4			126%	129%
5, 5, 5				141%
4, n, 4		175%		
5, n, 5		172%		
6, n, 6		169%	160%	139%
7, n, 7			155%	151%
8, n, 8			126%	200%
9, n, 9			183%	261%
10, n, 10			135%	186%
11, n, 11			186%	188%
12, n, 12			218%	226%
13, n, 13				203%
14, n, 14				268%
15, n, 15				208%
16, n, 16				207%
17, n, 17				285%
18, n, 18				274%
19, n, 19				273%
20, n, 20				329%

The leftmost column specifies how many existential guarantees were added of each type among A, B, and C, respectively

Fig. 2. Ratios between the running times of the configurations in SET2 and SET3, and the baseline configurations (vertical axis), as a function of the number of added existential guarantees (horizontal axis)

Moreover, interestingly, in 16 of the 82 configurations, the ratio was less than 100%, i.e., GR(1)* realizability checking was faster than that of GR(1). We may explain this phenomenon by observing that with additional existential guarantees, in each fixed-point iteration, a potentially larger set of states is removed from the set computed so far (consider the outer fixed-point operator in Eq. 4). Thus, adding existential guarantees may reduce the total number of fixed-point iterations and thus result in a faster realizability check.

Another notable observation is that GR(1)* realizability checking seems to scale well with the number and length of the added existential guarantees. Our experiments involved up to 45 existential guarantees, reaching a total length of 395 symbols. Figure 2 seems to suggest that the growth in running times is linear in the number of added existential guarantees, with a rather low slope.

Finally, to answer RQ2, the results for SET1, provided in Table 2, suggest no visible difference between the three types of existential guarantees we considered. Note however, that in comparison to RQ1, the data we obtained that is relevant to RQ2 is relatively small. Further experiments are required in order to provide a more reliable answer to RQ2.

11. Related work

Reactive synthesis for LTL was studied in [PR89] and shown to be 2EXPTIME-complete [Ros92]. The GR(1) fragment of LTL, which can be solved in time quadratic in the size of the state space, is proposed in [BJP⁺12]. As GR(1)* extends GR(1) with existential requirements, it is in fact a fragment of an extension of CTL* [EH86] with regular expressions. Thus, GR(1)* can be expressed in CTRL [MMDdJ11]. Kupferman and Vardi showed that the synthesis problem for CTL* formulas is 2EXPTIME-complete [KV99, KV00]. Recently, Bloem et al. suggested a CTL* synthesis technique [BSK17], and a corresponding synthesis tool. As we show, GR(1)*, like GR(1), is solved in time quadratic in the size of the state space.

Synthesis techniques that consider existential requirements, use cases, and scenarios were suggested in the literature. Harel et al. [HK02, KHP⁺05] studied synthesis of object systems from live sequence charts (LSCs). An LSC is an extension of a message sequence chart (MSC), which allows to specify universal and existential requirements. [KHP⁺05] presents an efficient translation of an LSC into a temporal logic formula, which enables to apply temporal logic based analysis and synthesis tools over LSC specification. [HK02] presents a synthesis technique for (universal and existential) LSCs. Maoz and Sa'ar [MS12] present synthesis from assume-guarantee scenarios in the form of LSC, by reduction to GR(1), and thus support only universal scenarios.

Uchithel et al. [SUB08, SBUK13, UBC07, UBC09] studied synthesis of a modal-transition-system (MTS) from LSCs. [SUB08] proposes an MTS synthesis algorithm from an existential LSC that triggers a universal LSC. The synthesized MTS fulfills two properties: (1) some of its behaviors fulfill the existential requirement; (2) whenever the existential requirement holds, the universal requirement must hold afterwards. [SBUK13] presents extensions and describes an implementation of that technique. [UBC07] and its extended version [UBC09] present an MTS synthesis algorithm from safety properties and scenarios.

Besides these papers, discussions about the value of use cases, scenarios, and examples for their use in the specification and analysis of systems can be found in [AKRU12, ARRU09, SMMM98, ZMT05]. All these motivated us to extend GR(1) with existential guarantees.

In the context of GR(1), Bloem et al. [BEK15] defined levels of cooperation between the system and the environment. Some of these levels of cooperation require that the justice assumptions hold in an existential manner. Ehlers et al. [EKB15] and Majumdar et al. [MPS19] proposed a synthesis technique for a cooperative GR(1) controller, i.e., a controller that never forces violation of the justice assumptions. Thus, while these papers relate to the assertions of the justice assumptions as existential guarantees, our technique allows to add regular expression as existential guarantees. Note that the problem solved in these papers is not a special case of our solution, since [EKB15] and [MPS19] require that the justice assumptions may hold from any reachable state, while we require that the existential guarantees can be satisfied along plays that satisfy the justice assumptions.

To the best of our knowledge, our work is the first to extend GR(1) with existential guarantees.

12. Conclusion

We introduced GR(1)*, which extends GR(1) with existential guarantees. GR(1)* allows engineers to add example use cases to the specification, so as to ensure that the synthesized controller will make them possible. We proved that the ‘exist’ operator allows to express properties that are inexpressible in LTL. Moreover, as we use regular expressions, GR(1)* formulas are inexpressible in CTL*.

We solved the realizability problem for GR(1)* specifications and presented a symbolic controller construction. Importantly, GR(1)* realizability check consumes $O((nm + \sum_{k=1}^{\ell} |reg_k|)N^2)$ -time where n is the number of justice guarantees, m is the number of justice assumptions, and $reg_1, \dots, reg_{\ell}$ are the regular expressions that appear in the existential guarantees. Hence, the time complexity of our realizability check is $O((|\varphi| + N)^2)$ where φ is the GR(1)* formula to be synthesized. This time complexity is asymptotically identical to the time complexity of the known GR(1) realizability check [BJP⁺12].

Intuitively, the existential operator we present, $GEF(reg)$, means that, globally, the play prefix can be extended to a play in which reg is satisfied along a segment that starts from a future state. We presented extensions to support additional forms of existential guaranties: globally, the play prefix can be extended to a play in which reg holds along a segment that (1) starts from the initial state and ends in a future state; (2) starts from some (past or future) state and ends in a future state; and (3) starts from some state and ends in some state. Moreover, we showed that the first form can be expressed using either regular expressions or past LTL formulas.

Further, we proved that if the environment is well-separated, our controller achieves a stronger requirement: from any reachable state there exists a run in which every regular expression that appears in an existential guarantee is satisfied infinitely often. More precisely, in the well-separated case, the existential guarantee $GEF(reg)$ can be replaced with the stronger guarantee $GEF((reg \text{ true}^*)^{\omega})$. As we proved, when the environment is well-separated, this observation allows one to support existential guaranties with DBW properties.

Finally, we discussed unrealizable GR(1)* games. The μ -calculus formula in Eq. 4 computes the winning region of the system player, but the negation of this formula does not necessarily compute the environment’s winning region. This holds since GR(1)* games may admit states from which neither players has a winning strategy.

We have implemented our synthesis technique, and integrated it in the Spectra GR(1) synthesizer [MR21, Spe]. To evaluate the efficiency of the suggested synthesis algorithm, we measured realizability checking times of baseline specifications without any existential guarantees and compared them to realizability checking times of the same specifications with different configurations of additional existential guarantees. The experiments revealed that our approach is efficient and scalable. The scalability of our work stems from its unique characteristics. Many proposed extensions of GR(1), e.g., [AKRV17, Ehl11, MPR16, MR15a], Sect. 5.2 in [BJP⁺12], and our extensions in Sect. 7, involve the addition of fresh auxiliary variables, which enlarges the state space and limits scalability. In contrast, our solution for GR(1)* does not enlarge the state space. Moreover, we construct NFA’s for the regular expressions with no need to determinize them. These two key factors enable the scalability of our work, as demonstrated in Fig. 2.

We consider the following future research directions. First, Firman et al. [FMR20] have recently presented several performance heuristics for GR(1) synthesis. These heuristics are transferred to realizability checking for GR(1)*, and we have used them in our experiments. It is interesting to look for additional heuristics that are tailored specifically for GR(1)* formulas.

Second, unrealizability is a well-known challenge for reactive synthesis in general and for GR(1) specifications in particular. Researchers have suggested to address GR(1) unrealizability using the concepts of unrealizable core, counter-strategy, and repair, see, e.g., [AMT13, CA17, CRST08, KHB13, KMR17, MRS19, MS13, MS21]. Since,

as we proved, unrealizable $\text{GR}(1)^*$ games do not always admit counter-strategies, it is interesting to investigate these concepts in the context of $\text{GR}(1)^*$.

Finally, we plan to implement and evaluate the extensions of $\text{GR}(1)^*$ we suggest in Sect. 7, including the support for past LTL existential guarantees and the operator $GE^+(reg)$ with its different suggested semantics.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Agreement No 638049, SYNTech).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

A. Proofs

Proof for Prop. 4.4. Take a variable y and consider the formula $A(GEF(\text{true}^*y))$. Note that this is indeed a $\text{GR}(1)^*$ formula, with no justice assumptions and guarantees. Towards a contradiction, assume that $A(GEF(\text{true}^*y))$ is equivalent to an LTL formula φ . Consider the following game structure,

$$GS = (\mathcal{X} = \{x\}, \mathcal{Y} = \{y\}, \theta^e = \neg x, \theta^s = \neg y, \rho^e = \text{true}, \rho^s = \text{true}).$$

Consider the strategy for the system f^s defined by $seq, \neg x \mapsto \neg y$ and $seq, x \mapsto y$, where $seq \in (2^{\mathcal{V}})^+$. That is, f^s assigns true to variable y iff the environment assigns true to x . It is easy to see that f^s is winning strategy w.r.t. the winning condition $A(GEF(\text{true}^*y))$.

Now, we change slightly the game structure by taking ρ^e to be the assertion $\neg x'$. That is, the environment always assigns false to x . Let π be a play consistent with f^s in the second game. Note that π is also a valid play in the first game structure thus $\pi \models \varphi$. Hence, f^s is also winning in the second game structure w.r.t. φ . On the other hand, in the second game f^s is clearly not winning w.r.t. $A(GEF(\text{true}^*y))$, which proves that φ is not equivalent to $A(GEF(\text{true}^*y))$. $\square$

Proof of Lem. 5.3. Before we turn to prove the lemma, we mention the seminal result that ω -regular games (and thus LTL games) are determined [GTW02]. That is, for every game structure $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, and an LTL formula φ , over $\mathcal{X} \cup \mathcal{Y}$, the winning regions of the system and the environment w.r.t. φ in GS form a partition of $2^{\mathcal{X} \cup \mathcal{Y}}$. We use this fact below.

$\llbracket \nu Z (V \vee (g(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z))) \rrbracket_{GS} = \bigcap_{t=0}^{\infty} Z_t$ where $Z_0 = 2^{\mathcal{X} \cup \mathcal{Y}}$ and $Z_{t+1} = \llbracket V \vee (g(Z_t) \wedge \bigwedge_{k=1}^{\ell} h_k(Z_t)) \rrbracket_{GS}$. We show, by induction, that the claim holds for each $s \in 2^{\mathcal{X} \cup \mathcal{Y}} \setminus Z_t$. This suffices since $Z_0 \supseteq Z_1 \supseteq \dots$ and thus, since the state space is finite, $\llbracket \nu Z (V \vee (g(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z))) \rrbracket_{GS} = Z_r$ for some sufficiently large r .

The claim vacuously holds for $t = 0$ thus we prove for $t + 1 > 0$, assuming that the induction hypothesis holds for Z_t . Take $s \notin Z_{t+1}$ and a strategy for the system f^s from s . We show that there is a play π , consistent with f^s , such that the pair (f^s, π) does not win for the system. Since $Z_{t+1} = \llbracket V \vee (g(Z_t) \wedge \bigwedge_{k=1}^{\ell} h_k(Z_t)) \rrbracket_{GS}$, $s \notin V$ and either $s \notin \llbracket \mu Y (\bigvee_{j=1}^m \nu X ((g_i \wedge \bigodot Z_t) \vee \bigodot Y \vee (\neg a_j \wedge \bigodot X))) \rrbracket_{GS}$ for some i , or $s \notin h_k(Z_t)$ for some k .

First, if $s \notin \llbracket \mu Y (\bigvee_{j=1}^m \nu X ((g_i \wedge \bigodot Z_t) \vee \bigodot Y \vee (\neg a_j \wedge \bigodot X))) \rrbracket_{GS}$, the claim follows from the correctness of Eq. 2. Specifically, by [KPP05, Lem. 9], the system cannot win from s w.r.t. the winning condition $(\bigvee_{j=1}^m FG(\neg a_j)) \vee F(g_i \wedge \bigodot Z_t)$. Hence, since ω -regular games are determined, the environment has a strategy to force from s the negation of the mentioned formula, which is equivalent to: $\bigwedge_{j=1}^m GF(a_j) \wedge G(\neg g_i \vee \bigodot \neg Z_t)$.

Therefore, the environment can force from s satisfaction of each assumption infinitely often without satisfying the justice guarantee g_i , or reaching a state $\hat{s} \in 2^{\mathcal{X} \cup \mathcal{Y}} \setminus Z_t$. If the former happens, the environment wins in that play, and if the latter happens, the induction hypothesis ensures that the prefix of that play can be extended to a play $\hat{\pi}$, consistent with f^s , such that the pair $(f^s, \hat{\pi})$ does not win for the system.

Now, in case that $s \notin h_k(Z_t)$, either $s \notin Z_t$ and we are done by the induction hypothesis, or $s \in Z_t$ but there is no path in Z_t from s that satisfies $\text{true}^* \text{reg}_k$. We assume that the latter occurs, and distinguish between two possibilities. First, suppose that every play from s , consistent with f^s , never leaves Z_t . Recall that $s \notin V$. Hence, the determinacy of ω -regular games ensures that the environment has a strategy to force satisfaction of the formula $\bigwedge_{j=1}^n GF(a_j)$ from s . Let f^e be such a strategy, and let π be the (unique) play from s , consistent with f^s and f^e . Consider a prefix, $\pi(0), \dots, \pi(i)$, of π . By assumption, every extension of this prefix to a play $\hat{\pi}$, consistent with f^s , never leaves Z_t . Therefore, for every $j \geq i$, $\hat{\pi}(i) \cdots \hat{\pi}(j) \notin L(\text{true}^* \text{reg}_k)$, which proves that (f^s, π) does not win for the system.

Now, suppose that there exists a play π , consistent with f^s , such that $\pi(i) \notin Z_t$. Then, by the induction hypothesis, the prefix $\pi(0), \dots, \pi(i)$ can be extended into a play $\hat{\pi}$, such that the pair $(f^s, \hat{\pi})$ does not win for the system. $\square$

Proof of Lem. 6.1. Take $s \in V'$, and assume, towards a contradiction, that $s \notin V$. Since $W = V \cup (\llbracket g \rrbracket(W)) \cap \bigcap_{k=1}^{\ell} h_k(W)$, $s \in \bigcap_{k=1}^{\ell} h_k(W)$, in contradiction to $s \in V'$. $\square$

Proof of Theorem 6.2. We slightly modify Alg. 2, and prove the correctness of the modified algorithm. Then, we will show that the modification is redundant, which implies the correctness of Alg. 2.

Algorithm 3 is obtained from Alg. 2 by the addition of new instructions, marked by the symbol '+'. Consider line 7 of Alg. 2, in which the play fulfills the last justice guarantee, and reaches a state $s \models g_n \wedge \odot W$. If $s \in h_{\text{next_ex}}(W)$ (the test in line 8), the controller switches to phase II. Otherwise, we presume that $s \in W \setminus \bigcap_{k=1}^{\ell} h_k(W) = V'$, and the controller will activate C_V for the remainder of the play. Apparently, it is also possible that $s \in \odot W \setminus W$. To handle this case, in Alg. 3, we add a Boolean variable *correction*, initialized to *false* (line 2). Before switching to phase II, we check if the next state is in $\odot W \setminus W$ (line 17), and if so, we turn on *correction*, to signal that a correction is needed before switching phase. When *correction* stores *true*, the controller takes an intermediate step that leads the play into W (line 2), and checks if this next state is in $h_{\text{next_ex}}(W)$ (line 3). If it is in $h_{\text{next_ex}}(W)$, the controller switches to phase II (line 5), and otherwise it stays in phase I.

First, we show that the construction in Alg. 3 defines a properly working GR(1) controller. That is, we show that the controller successfully replies with legal outputs to any sequence of legal inputs by the environment. Afterwards, we prove the correctness of the construction in Alg. 3. Then, finally, we will conclude that $\odot W \subseteq W$, which implies that the modification is redundant, as the condition in line 17 never holds. Consequently, the correctness of Alg. 2 follows. $\square$

Algorithm 3 Modified GR(1) controller

```

/*      Initialization      */
input:  $s_x \models \theta^e$ 
1:  $(next\_jus, next\_ex, mem, phase) \leftarrow (1, 1, 1, 1)$ 
2:  $correction \leftarrow \text{false}$  // +
3: return  $s_y$ , such that  $(s_x, s_y) \models (\theta^s \wedge W)$ 

/*      From state  $s$       */
input:  $s_x$  such that  $(s, s'_x) \models \rho^e$ 
1: if  $correction$  then // +
2:   choose  $s_y$  such that  $(s_x, s_y) \in W$  // +
3:   if there exists  $i$  such that // +
     •  $(q_i^{next\_ex}, (s_x, s_y)) \in R_{(next\_ex, W)}(\mathcal{F}^k \times W)$ , and // +
     •  $q_i^{next\_ex} \in \Delta^{next\_ex}(q_0, (s_x, s_y))$  then // +
4:      $mem \leftarrow i$  as above // +
5:      $phase \leftarrow 2$  // +
6:   end if // +
7:    $correction \leftarrow \text{false}$  // +
8:   return  $s_y$  // +
9: end if
10: if  $phase == 1$  then // Phase I: Fulfilling a justice guarantee
11:   if  $s \models V'$  then return  $C_V(s, s_x)$  // The play reached  $V'$ 
12:   end if
13:    $s_y \leftarrow C_{(GR(1), next\_jus)}(s, s_x)$ 
14:   if  $((s_x, s_y) \models g_{next\_jus} \wedge \bigotimes W) \wedge (next\_jus < n)$  then  $next\_jus \leftarrow next\_jus + 1$ 
15:   end if
16:   if  $((s_x, s_y) \models g_{next\_jus} \wedge \bigotimes W) \wedge (next\_jus = n)$  then
17:     if  $(s_x, s_y) \in \bigotimes W \setminus W$  then // +
18:        $correction \leftarrow \text{true}$  // +
19:       return  $s_y$  // +
20:     end if // +
21:     if there exists  $i$  such that
       •  $(q_i^{next\_ex}, (s_x, s_y)) \in R_{(next\_ex, W)}(\mathcal{F}^k \times W)$ , and
       •  $q_i^{next\_ex} \in \Delta^{next\_ex}(q_0, (s_x, s_y))$  then
22:        $mem \leftarrow i$  as above
23:        $phase \leftarrow 2$ 
24:     end if
25:   end if
26:   return  $s_y$ 
27: end if
28: if  $phase == 2$  then // Phase II: Fulfilling an existential guarantee
29:    $(s_y, mem) \leftarrow C_{next\_ex}(s, s_x, mem)$ 
30:   if  $s_y \neq \text{"undefined"}$  then return  $s_y$ 
31:   end if
32:    $(next\_jus, next\_ex, phase) \leftarrow (1, next\_ex \oplus 1, 1)$ 
33:   goto line 1
34: end if

```

Let C be the controller depicted in Alg. 3. First, we argue that C is a properly defined controller. That is, if $s_0 \in W$, $s_0, \dots, s_i$ is a sequence of states, consistent with C , and $(s_i, s_x) \models \rho^e$, then there exists $s_y \in 2^{\mathcal{Y}}$ such that $C(s_0, \dots, s_i, s_x) = s_y$. Note that this also implies that a play from W that is consistent with C never reaches a deadlock for the system.

To conclude this fact, we prove a useful statement.

Claim 1. $W \cup \bigotimes W \subseteq \llbracket g(W) \rrbracket_{GS}$.

Proof of Claim 1. For each justice guarantee g_i let $U_i = \llbracket \mu Y (\bigvee_{j=1}^m \nu X ((g_i \wedge \otimes W) \vee \otimes Y \vee (\neg a_j \wedge \otimes X))) \rrbracket_{GS}$. Hence, $\llbracket g(W) \rrbracket_{GS} = \bigcap_{i=1}^n U_i$. By [KPP05, Lem. 9], U_i is the system's winning region for the game whose winning condition is $(\bigvee_{j=1}^m FG(\neg a_j)) \vee F(g_i \wedge \otimes W)$. Since V is the system's winning region for the game whose winning condition is $\bigvee_{j=1}^m FG(\neg a_j)$, $V \subseteq U_i$ for each $1 \leq i \leq n$. Hence, $V \subseteq \bigcap_{i=1}^n U_i = \llbracket g(W) \rrbracket_{GS}$.

Now, W is a fixed-point of the formula $V \vee (g(W) \wedge \bigwedge_{k=1}^\ell h_k(W))$. Namely, $W = \llbracket V \vee (g(W) \wedge \bigwedge_{k=1}^\ell h_k(W)) \rrbracket_{GS} = V \cup (\llbracket g(W) \rrbracket_{GS} \cap \bigcap_{k=1}^\ell \llbracket h_k(W) \rrbracket_{GS})$. Since $V \subseteq \llbracket g(W) \rrbracket_{GS}$, $W = \llbracket g(W) \rrbracket_{GS} \cap \bigcap_{k=1}^\ell \llbracket h_k(W) \vee V \rrbracket_{GS}$ thus $W \subseteq \llbracket g(W) \rrbracket_{GS}$. Moreover, it follows that $W \subseteq U_i$ for each $1 \leq i \leq n$. Therefore, by definition of U_i , also $\otimes W \subseteq U_i$, and we get that $\otimes W \subseteq \llbracket g(W) \rrbracket_{GS}$. $\square$

Now, we show, in guidelines, that the controller C is indeed well-defined, and from every reachable state and a legal input it successfully computes a legal output. We leave it to the reader to verify that this statement is implied by the following observations, in a straightforward manner.

1. The play begins from a state $s \in W$ (initialization, line 3). Thus, by Claim 1, the play begins from a state $s \in \llbracket g(W) \rrbracket_{GS}$, from which $C_{(GR(1),1)}$ can be activated.
2. If the condition in line 11 holds, by Lem. 6.1, C_V can be activated, and it returns a state in $V \subseteq W$.
3. If at the computation from state s , C increases $next_jus$ by 1 (line 14), then the ensuing state (s_x, s_y) satisfies $(s_x, s_y) \in \otimes W$. Hence, by Claim 1, $(s_x, s_y) \in \llbracket g(W) \rrbracket_{GS}$, and $C_{(GR(1),next_jus+1)}$ can operate in the next state.
4. When the play traverses from phase I to phase II (line 23), the ensuing state (s_x, s_y) satisfies $(s_x, s_y) \in W$. Thus, each C_k can be activated.
5. Since a computation that is consistent with C_k never leaves W , when C switches from phase II to phase I (line 32), the current state is in W . Hence, by Claim 1, $C_{(GR(1),1)}$ can be activated.

Now, finally, we turn to prove that C wins w.r.t. the GR(1)* winning condition $\psi = A(\bigwedge_{j=1}^m GF(a_j) \rightarrow (\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k)))$. First, we argue that once C_V is activated (line 11), it will be activated for the remainder of the play, and thus the controller will force violation of the justice assumptions. This follows by the next claim.

Claim 2. Take $s \in V'$ and $s_x \in 2^X$ such that $(s, s_x) \models \rho^e$. For $s_y = C_V(s, s_x)$, $(s_x, s_y) \in V'$.

Proof of Claim 2. Assume towards a contradiction that $(s_x, s_y) \notin V'$. Hence, $(s_x, s_y) \in V \setminus V'$, and thus $(s_x, s_y) \in W \setminus V'$. Therefore, $(s_x, s_y) \in \bigcap_{k=1}^\ell h_k(W)$ and hence, for every $k \in \{1, \dots, \ell\}$ we have $(s_x, s_y) \in h_k(W)$. That is, for every $k \in \{1, \dots, \ell\}$ there exists a path in W , $s_0, \dots, s_{m(k)}$, such that $s_0 = (s_x, s_y)$, $(s_i, s_{i+1}) \models \rho^e \wedge \rho^s$ for each $0 \leq i \leq m(k) - 1$, and $s_0, \dots, s_{m(k)} \in L(\text{true}^* reg_k)$. Consequently, since $(s, (s_x, s_y)) \models \rho^e \wedge \rho^s$, we get that the same holds for $s, s_0, \dots, s_{m(k)}$. Hence, for every $k \in \{1, \dots, \ell\}$ we have $s \in h_k(W)$. Therefore, $s \in \bigcap_{k=1}^\ell h_k(W)$, in contradiction to $s \in V' = W \setminus \bigcap_{k=1}^\ell h_k(W)$. $\square$

Now, consider a play π , consistent with C , and we shall prove that π wins from the system. If the justice assumptions are violated in π , then we are done. Otherwise, C never activates C_V . First, we show that C switches from phase I to phase II infinitely many times. Assume towards a contradiction that this claim is false. Note that the play cannot stay in phase II forever thus, from some state $\pi(i)$, $phase$ stores 1 permanently. Since $\pi^i \models \bigwedge_{i=1}^m GF(a_i)$, $next_jus$ is increased in the play-suffix π^i until it stores n , and afterwards, for some state $\pi(i+j)$, the computation of C reaches line 16, and the condition in that line holds. Now, we concentrate on the successor state $(s_x, s_y) = \pi(i+j+1)$. If $\pi(i+j+1) \in W$, then the controller will reach line 21, and if $\pi(i+j+1) \in \otimes W \setminus W$, then the controller will reach line 3 in the computation from state $\pi(i+j+1)$.

Assume, w.l.o.g., that the former occurs. That is, $\pi(i+j+1) \in W$ and thus the computation from $\pi(i+j)$ reaches line 21. Since, by assumption, the play never reaches phase II in π^{i+j} , the condition in line 21 does not hold for the ensuing state $\pi(i+j+1)$. Hence, for every path p in W from $\pi(i+j+1)$ that is consistent with $\rho^e \wedge \rho^s$, $p \notin L(\text{true}^* reg_{next_ex})$, and thus $\pi(i+j+1) \notin h_{next_ex}(W)$. As a result, $\pi(i+j+1) \in V'$ and C activates C_V in the computation from $\pi(i+j+1)$. Therefore, by Claim 2, the justice assumptions are violated in π , in contradiction to a former assumption.

Now, we proved that the controller switches to phase II in π infinitely many times. Hence, each controller $C_{(GR(1),i)}$ is activated infinitely many times, and in each activation $C_{(GR(1),i)}$ eventually reaches a state in $g_i \wedge \otimes W$. Therefore, $\pi \models \bigwedge_{i=1}^n GF(g_i)$. Further, for every state $\pi(i)$ in π and every $k \leq \ell$, there exists a state $\pi(t)$ with $t > i$ such that in $\pi(t)$, C switches to phase II, and $next_ex$ stores k . Hence, C_k is activated from a state that follows $\pi(t)$ until it returns: “undefined”. Therefore, there exists a play $\hat{\pi}$, consistent with C , that starts with the prefix $\pi(0), \dots, \pi(t)$, and for some $r, \hat{r}$ with $t < r \leq \hat{r}$, we have $\hat{\pi}(r) \cdots \hat{\pi}(\hat{r}) \models reg_k$, as required.

Now, finally, we show that what we have proved implies the correctness of Alg. 2. We proved that the GR(1) controller constructed in Alg. 3 wins the GR(1)* game from the set W , computed by Eq. 4. Hence, the system has a winning strategy from $\odot W$: it takes a single step to reach W , and then activates the GR(1) controller of Alg. 3 for the remainder of the play. But, by Lem. 5.3, the system has no winning strategy from any state $s \in 2^{\mathcal{X} \cup \mathcal{Y}} \setminus W$. Hence, $\odot W \subseteq W$. Consequently, line 17 in Alg. 3 deals with a vacuous situation and it is redundant. The correctness of Alg. 2 follows.

Proof of Lem. 8.3. The “if” direction is simple since ψ implies ϕ . For the other direction, assume that f^s wins from s w.r.t. ϕ in $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$. We show that it also wins from s w.r.t. ψ in GS .

Take a play π , consistent with f^s . We need to show that the requirements of Def. 8.2 hold. First, since f^s wins from s w.r.t. ϕ in GS , π does not end in a deadlock for the system. Moreover, since the environment is well-separated, there are no deadlocks for the environment. Hence, π is infinite and the first item of Def. 8.2 holds. Furthermore, clearly, the second item holds since the pair (f^s, π) wins for the system w.r.t. ϕ in GS .

It is left to show that item 3 of Def. 8.2 holds as well. Take a regular expression reg_k , and $i \geq 0$. It suffices to show that there exist a play $\hat{\pi}$, consistent with f^s , such that $(\pi(0), \dots, \pi(i)) = (\hat{\pi}(0), \dots, \hat{\pi}(i))$, and for some $j \geq i$, it holds that $\hat{\pi}(i) \dots \hat{\pi}(j) \in L(\text{true}^* reg_k)$. This is indeed sufficient since, by repeatedly invoking this claim, one can construct a play $\tilde{\pi}$, consistent with f^s , such that $(\pi(0), \dots, \pi(i)) = (\tilde{\pi}(0), \dots, \tilde{\pi}(i))$ and $\tilde{\pi}^i \in L(\text{true}^*(reg_k \text{true}^*)^\omega)$.

Consider a prefix of $\pi, \pi(0), \dots, \pi(i)$. Since the pair $(GS, \{a_1, \dots, a_m\})$ has a well-separated environment, the environment can force from $\pi(i)$ satisfaction of the justice assumptions infinitely often. Therefore, there exists a play $\tilde{\pi}$, consistent with f^s , such that $(\pi(0), \dots, \pi(i)) = (\tilde{\pi}(0), \dots, \tilde{\pi}(i))$, and $\tilde{\pi} \models \bigwedge_{j=1}^m GF(a_j)$. Hence, since f^s wins from s w.r.t. ϕ in GS , there exists a play $\hat{\pi}$, consistent with f^s , such that $(\pi(0), \dots, \pi(i)) = (\hat{\pi}(0), \dots, \hat{\pi}(i))$, and for some $j \geq i$, $\hat{\pi}(i) \dots \hat{\pi}(j) \in L(\text{true}^* reg_k)$, as required. $\square$

Proof of Lem. 8.4. Since, by the construction, each auxiliary variable y_t tracks the state of the DBW $\mathcal{A}_t$, it is not difficult to see that the ‘only if’ direction holds. Hence, we prove solely the ‘if’ direction. Take a state s such that for each DBW property ψ_t , the value of y_t in s is q_0^t , the initial state of the DBW $\mathcal{A}_t$. Assume that the strategy f^s wins from s w.r.t. $\psi = A(\bigwedge_{j=1}^m GF(a_j) \rightarrow (\bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF(reg_k) \wedge \bigwedge_{t=1}^r GEF(y_t \in \mathcal{F}^t)))$ in $\tilde{GS}$. We claim that f^s wins from $s \mid_{\mathcal{X} \cup \mathcal{Y}}$ w.r.t. $\phi = A(\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i) \wedge \bigwedge_{k=1}^\ell GEF((reg_k \text{true}^*)^\omega) \wedge \bigwedge_{t=1}^r GE^+(\psi_t))$ in GS .

Since the environment is well separated, by Lem. 8.3, f^s wins in from s in $\tilde{GS}$ w.r.t.

$$A((\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)) \wedge \bigwedge_{k=1}^\ell GEF((reg_k \text{true}^*)^\omega) \wedge \bigwedge_{t=1}^r GEF(((y_t \in \mathcal{F}^t) \text{true}^*)^\omega)). \quad (5)$$

Therefore, f^s wins from $s \mid_{\mathcal{X} \cup \mathcal{Y}}$ w.r.t. $A((\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)) \wedge \bigwedge_{k=1}^\ell GEF((reg_k \text{true}^*)^\omega))$ in GS . It is left to show that the requirement $\bigwedge_{t=1}^r GE^+(\psi_t)$ holds as well, for every play from $s \mid_{\mathcal{X} \cup \mathcal{Y}}$, consistent with f^s .

Let π be a play in GS from $s \mid_{\mathcal{X} \cup \mathcal{Y}}$, consistent with f^s , and let $\tilde{\pi}$ be the unique play in $\tilde{GS}$ from s , consistent with f^s , such that $\tilde{\pi}(i) \mid_{\mathcal{X} \cup \mathcal{Y}} = \pi(i)$. By Eq. 5, it holds that for each $t \leq r$, any prefix of $\tilde{\pi}, \tilde{\pi}(0), \dots, \tilde{\pi}(i)$, can be extended to a play $\tilde{\pi}_1$ in which the assertion $y_t \in \mathcal{F}^t$ holds infinitely often. Consequently, the play π_1 , defined by $\pi_1(j) = \tilde{\pi}_1(j) \mid_{\mathcal{X} \cup \mathcal{Y}}$, extends the play-prefix $\pi(0), \dots, \pi(i)$. By the construction, $\pi_1 \models \psi_t$ and thus f^s wins from $s \mid_{\mathcal{X} \cup \mathcal{Y}}$ w.r.t. $\phi = A((\bigwedge_{j=1}^m GF(a_j) \rightarrow \bigwedge_{i=1}^n GF(g_i)) \wedge \bigwedge_{k=1}^\ell GEF((reg_k \text{true}^*)^\omega) \wedge \bigwedge_{t=1}^r GE^+(\psi_t))$ in GS , as required. $\square$

proof of Lem. 9.3. Let f^s be a strategy for the system from s . We show that there exists a play π , consistent with f^s , such that the pair (f^s, π) does not win for the system.

First, suppose that any play, consistent with f^s , never reaches a state $t \models q$. In this case, the existential guarantee $GEF(q)$ clearly does not hold in any play, consistent with f^s . Therefore, (f^s, π) does not win for the system for any play π , consistent with f^s .

Now, assume that π is consistent with f^s , and $\pi(i) \models q$. Hence, for every $j > i$, $\pi(j) \models \neg q$. Therefore, the pair (f^s, π) does not win for the system since it violates the justice guarantee $GF(q)$. $\square$

proof of Lem. 9.4. We construct a strategy for the system f^s such that the (unique) play, $\pi = \pi(0), \pi(1), \dots$ from s that is consistent with f^s and f^e satisfies:

- For every $i > 0$, $\pi(i) \models g \wedge \neg q$.
- The pair (f^s, π) wins for the system.

We construct f^s inductively by $f^s = \bigcup_{i=1}^{\infty} f_i^s$, where $f_i^s : (2^{\mathcal{X} \cup \mathcal{Y}})^i \times 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$. Assume that $k > 0$, and the functions $\{f_i^s : i < k\}$ have been defined, so that the only prefix of a play form $s, \pi(0), \dots, \pi(k)$, that is consistent with f^e and $\bigcup_{i=1}^{k-1} f_i^s$, satisfies $\forall i \in \{1, \dots, k\}, \pi(i) \models g \wedge \neg q$. Consider the value $f^e(\pi(0), \dots, \pi(k))$. This value is either $f^e(\pi(0), \dots, \pi(k)) = \{x\}$, or $f^e(\pi(0), \dots, \pi(k)) = \emptyset$. If $f^e(\pi(0), \dots, \pi(k)) = \{x\}$ we set $f_k^s(\pi(0), \dots, \pi(k), \{x\}) = \{g\}$, and $f_k^s(\pi(0), \dots, \pi(k), \emptyset) = \{q\}$. Otherwise, we set the opposite: $f_k^s(\pi(0), \dots, \pi(k), \emptyset) = \{g\}$, and $f_k^s(\pi(0), \dots, \pi(k), \{x\}) = \{q\}$.

In words, to define the function f_k^s , we consider the next input of f^e . f_k^s is constructed so that it will reply with $\{g\}$ to the expected input by f^e , and with $\{q\}$ to the input that f^e is not about to provide. Note that this definition is valid. Indeed, if $k > 0$, then $\pi(k) \models g \wedge \neg q$ thus the output $\{g\}$ is allowed. Likewise, if $k = 0$, then $\pi(k) = \pi(0) = s$, and $s \models \neg q$ thus, again, $\{g\}$ is a legal output. Further, as required, for the prefix $\pi(0), \dots, \pi(k+1)$ it also holds that $\pi(k+1) \models g \wedge \neg q$.

It remains to show that (f^s, π) wins for the system, where π is the play consistent with f^s and f^e . Note that by the construction, $\pi(i) \models g \wedge \neg q$ for every $i > 0$. Hence, $\pi \models GF(g)$. Now, consider a prefix of π , $\pi(0), \dots, \pi(k)$. Assume, without loss of generality, that $f^e(\pi(0), \dots, \pi(k)) = \{x\}$. Hence, by the construction, $f^s(\pi(0), \dots, \pi(k), \emptyset) = \{q\}$. Therefore, $\pi(0), \dots, \pi(k)$ can be extended into a play $\hat{\pi}$ such that $\hat{\pi}(k+1) \models q$. Consequently, the pair (f^s, π) wins for the system. $\square$

References

- [AKRU12] Alrajeh D, Kramer J, Russo A, Uchitel S (2012) Learning from vacuously satisfiable scenario-based specifications. In: International conference on fundamental approaches to software engineering. Springer, pp 377–393
- [AKRV17] Almagor S, Kupferman O, Ringert JO, Verner Y (2017) Quantitative assume guarantee synthesis. In: Rupak M, Viktor K (eds) Computer aided verification—29th international conference, CAV 2017, Heidelberg, Germany, July 24–28, 2017, Proceedings, Part II, volume 10427 of lecture notes in computer science. Springer, pp 353–374
- [AM04] Alexander IF, Maiden N (2004) Scenarios, stories, use cases: through the systems development life-cycle, 1st ed. Wiley
- [AMP19] Amram G, Maoz S, Pistiner O (2019) GR(1)*: GR(1) specifications extended with existential guarantees. In: ter Beek MH, McIver A, Oliveira JN (eds) Formal methods—the next 30 years—third world congress, FM 2019, Porto, Portugal, October 7–11, 2019, Proceedings, volume 11800 of lecture notes in computer science. Springer, pp 83–100
- [AMT13] Alur R, Moarref S, Topcu U (2013) Counter-strategy guided refinement of GR(1) temporal logic specifications. In: Formal methods in computer-aided design, FMCAD 2013, Portland, OR, USA, October 20–23, 2013. IEEE, pp 26–33
- [ARM] AMBA AHB specification. <http://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications>
- [ARRU09] Alrajeh D, Ray O, Russo A, Uchitel S (2009) Using abduction and induction for operational requirements elaboration. J Appl Logic, 7(3):275–288
- [BCJ⁺97] Browne A, Clarke EM, Jha S, Long DE, Marrero WR (1997) An improved algorithm for the evaluation of fixpoint expressions. Theor Comput Sci 178(1–2):237–255
- [BEK15] Bloem R, Ehlers R, Könighofer R (2015) Cooperative reactive synthesis. In: Finkbeiner B, Pu G, Zhang L (eds) Automated technology for verification and analysis—13th international symposium, ATVA 2015, Shanghai, China, October 12–15, 2015, Proceedings, volume 9364 of lecture notes in computer science. Springer, pp 394–410
- [BGJ⁺07] Bloem R, Galler SJ, Jobstmann B, Piterman N, Pnueli A, Weiglhofer M (2007) Interactive presentation: automatic hardware synthesis from specifications: a case study. In: 2007 Design, automation and test in Europe conference and exposition, DATE 2007, Nice, France, April 16–20, 2007, pp 1188–1193
- [BJK14] Bloem R, Jacobs S, Khalimov A (2014) Parameterized synthesis case study: AMBA AHB. In: Chatterjee K, Ehlers R, Jha S (eds) Proceedings 3rd workshop on synthesis, SYNT 2014, Vienna, Austria, July 23–24, 2014, volume 157 of EPTCS, pp 68–83
- [BJP⁺12] Bloem R, Jobstmann B, Piterman N, Pnueli A, Sa’ar Y (2012) Synthesis of reactive(1) designs. J Comput Syst Sci 78(3):911–938
- [Bry86] Bryant RE (1986) Graph-based algorithms for boolean function manipulation. IEEE Trans Comput 35(8):677–691
- [BSK17] Bloem R, Schewe S, Khalimov A (2017) CTL* synthesis via LTL synthesis. In: Proceedings sixth workshop on synthesis, SYNT@CAV 2017, Heidelberg, Germany, 22nd July 2017, pp 4–22
- [CA17] Cavezza DG, Alrajeh D (2017) Interpolation-based GR(1) assumptions refinement. In: TACAS, volume 10205 of LNCS, pp 281–297
- [CE81] Clarke EM, Emerson EA (1981) Design and synthesis of synchronization skeletons using branching time temporal logic. In: Workshop on logic of programs. Springer, pp 52–71
- [CRST08] Cimatti A, Roveri M, Schuppan V, Tchaltsev A (2008) Diagnostic information for realizability. In: VMCAI, volume 4905 of LNCS. Springer, pp 52–67
- [DAC99] Dwyer MB, Avrunin GS, Corbett JC (1999) Patterns in property specifications for finite-state verification. In: ICSE. ACM, pp 411–420
- [DBPU13] D’Ippolito N, Braberman VA, Piterman N, Uchitel S (2013) Synthesizing nonanomalous event-based controllers for liveness goals. ACM Trans Softw Eng Methodol 22(1):9
- [EH86] Emerson AE, Halpern JY (1986) “sometimes” and “not never” revisited: on branching versus linear time temporal logic. J ACM 33(1):151–178
- [Ehl11] Ehlers R (2011) Generalized Rabin(1) synthesis with applications to robust system synthesis. In: NASA formal methods, volume 6617 of LNCS. Springer, pp 101–115

- [EKB15] Ehlers R, Könighofer R, Bloem R (2015) Synthesizing cooperative reactive mission plans. In: 2015 IEEE/RSJ international conference on intelligent robots and systems, IROS 2015, Hamburg, Germany, September 28–October 2, 2015. IEEE, pp 3478–3485
- [FDL⁺16] Filippidis I, Dathathri S, Livingston SC, Ozay N, Murray RM (2016) Control design for hybrid systems with tulip: the temporal logic planning toolbox. In: 2016 IEEE conference on control applications, CCA 2016, Buenos Aires, Argentina, September 19–22, 2016. IEEE, pp 1030–1041
- [FMR20] Firman E, Maoz S, Ringert JO (2020) Performance heuristics for GR(1) synthesis and related algorithms. *Acta Inform* 57(1–2):37–79
- [GCH13] Godhal Y, Chatterjee K, Henzinger TA (2013) Synthesis of AMBA AHB from formal specification: a case study. *STTT* 15(5–6):585–601
- [GTW02] Grädel E, Thomas W, Wilke T (2002) editors. Automata, logics, and infinite games: a guide to current research [outcome of a Dagstuhl seminar, February 2001], volume 2500 of lecture notes in computer science. Springer
- [HK02] Harel D, Kugler H (2002) Synthesizing state-based object systems from LSC specifications. *Int J Found Comput Sci* 13(01):5–51
- [Hop08] Hopcroft JE (2008) Introduction to automata theory, languages, and computation. Pearson Education, India
- [Jac04] Jacobson I (2004) Object-oriented software engineering: a use case driven approach. Addison Wesley, Redwood City
- [KFP09] Kress-Gazit H, Fainekos GE, Pappas GJ (2009) Temporal-logic-based reactive mission and motion planning. *IEEE Trans Robot* 25(6):1370–1381
- [KHB13] Könighofer R, Hofferek G, Bloem R (2013) Debugging formal specifications: a practical approach using model-based diagnosis and counterstrategies. *STTT* 15(5–6):563–583
- [KHP⁺05] Kugler H, Harel D, Pnueli A, Lu Y, Bontemps Y Temporal logic for scenario-based specifications. In: International conference on tools and algorithms for the construction and analysis of systems. Springer, pp 445–460
- [KMR17] Kuvent A, Maoz S, Ringert JO (2017) A symbolic justice violations transition system for unrealizable GR(1) specifications. In: Bodden E, Schäfer W, van Deursen A, Zisman A (eds) Proceedings of the 2017 11th joint meeting on foundations of software engineering, ESEC/FSE 2017, Paderborn, Germany, September 4–8, 2017. ACM, pp 362–372
- [Koz82] Kozen D (1982) Results on the propositional μ -calculus. In: Proceedings of the 9th colloquium on automata, languages and programming. Springer, London, pp 348–359
- [KP10] Klein U, Pnueli A (2010) Revisiting synthesis of GR(1) specifications. In: Barner S, Harris S, Kroening D, Raz O (eds) Hardware and software: verification and testing—6th international Haifa verification conference, HVC 2010, Haifa, Israel, October 4–7, 2010. Revised selected papers, volume 6504 of lecture notes in computer science. Springer, pp 161–181
- [KPP05] Kesten Y, Piterman N, Pnueli A (2005) Bridging the gap between fair simulation and trace inclusion. *Inf Comput* 200(1):35–61
- [KPR98] Kesten Y, Pnueli A, Raviv L (1998) Algorithmic verification of linear temporal logic specifications. In: International colloquium on automata, languages, and programming. Springer, pp 1–16
- [KV99] Kupferman O, Vardi MY (1999) Church's problem revisited. *Bull Symb Logic* 5(2):245–263
- [KV00] Kupferman O, Vardi MY (2000) Synthesis with incomplete informatio. In: Advances in temporal logic. Springer, pp 109–127
- [MMDdJ11] Mateescu R, Monteiro PT, Dumas E, de Jong H (2011) CTRL: extension of CTL with regular expressions and fairness operators to verify genetic regulatory networks. *Theor Comput Sci* 412(26):2854–2883
- [MPR16] Maoz S, Pistiner O, Ringert JO (2016) Symbolic BDD and ADD algorithms for energy games. In: Piskac R, Dimitrova R (eds) Proceedings fifth workshop on synthesis, SYNT@CAV 2016, Toronto, Canada, July 17–18, 2016., volume 229 of EPTCS, pp 35–54
- [MPS19] Majumdar R, Piterman N, Schmuck A-K (2019) Environmentally-friendly GR(1) synthesis. *CoRR*, [arXiv:1902.05629](https://arxiv.org/abs/1902.05629)
- [MR15a] Maoz S, Ringert JO (2015) GR(1) synthesis for LTL specification patterns. In: ESEC/FSE. ACM, pp 96–106
- [MR15b] Maoz S, Ringert JO (2015) Synthesizing a Lego Forklift controller in GR(1): a case study. In: Proceedings of the 4th workshop on synthesis, SYNT 2015 colocated with CAV 2015, volume 202 of EPTCS, pp 58–72
- [MR16] Maoz S, Ringert JO (2016) On well-separation of GR(1) specifications. In: Zimmermann T, Cleland-Huang J, Su Z (eds) Proceedings of the 24th ACM SIGSOFT international symposium on foundations of software engineering, FSE 2016, Seattle, WA, USA, November 13–18, 2016. ACM, pp 362–372
- [MR21] Maoz S, Ringert JO (2021) Spectra: a specification language for reactive systems. *Softw Syst Model*. <https://doi.org/10.1007/s10270-021-00868-z>
- [MRS19] Maoz S, Ringert JO, Shalom R (2019) Symbolic repairs for GR(1) specifications. In: Atlee JM, Bultan T, Whittle J (eds) Proceedings of the 41st international conference on software engineering, ICSE 2019, Montreal, QC, Canada, May 25–31, 2019. IEEE/ACM, pp 1016–1026
- [MS11] Maoz S, Sa'ar Y (2011) AspectLTL: an aspect language for LTL specifications. In: AOSD. ACM, pp 19–30
- [MS12] Maoz S, Sa'ar Y (2012) Assume-guarantee scenarios: semantics and synthesis. In: MODELS, volume 7590 of LNCS. Springer, pp 335–351
- [MS13] Maoz S, Sa'ar Y (2013) Counter play-out: executing unrealizable scenario-based specifications. In: ICSE. IEEE/ACM, pp 242–251
- [MS20a] Maoz S, Shalom R (2020) Inherent vacuity for GR(1) specifications. In: ESEC/FSE. ACM, pp 99–110
- [MS20b] Maoz S, Shevrin I (2020) Just-in-time reactive synthesis. In: ASE. ACM, pp 635–646
- [MS21] Maoz S, Shalom R (2021) Unrealizable cores for reactive systems specifications. In: ICSE (to appear)
- [MSP⁺16] Maniatopoulos S, Schillinger P, Pong V, Conner DC, Kress-Gazit H (2016) Reactive high-level behavior synthesis for an atlas humanoid robot. In: Kragic D, Bichi A, De Luca A (eds) 2016 IEEE international conference on robotics and automation, ICRA 2016, Stockholm, Sweden, May 16–21, 2016. IEEE, pp 4192–4199
- [Pnu77] Pnueli A (1977) The temporal logic of programs. In: 18th Annual symposium on foundations of computer science, Providence, Rhode Island, USA, 31 October–1 November 1977. IEEE Computer Society, pp 46–57
- [Poh10] Pohl K (2010) Requirements engineering: fundamentals, principles, and techniques, 1st ed. Springer
- [PPS06] Piterman N, Pnueli A, Sa'ar Y (2006) Synthesis of reactive(1) designs. In: VMCAI, volume 3855 of LNCS. Springer, pp 364–380
- [PR89] Pnueli A, Rosner R (1989) On the synthesis of a reactive module. In: POPL. ACM Press, pp 179–190

- [Ros92] Rosner R (1992) Modular synthesis of reactive systems. PhD thesis, Weizmann Institute of Science
- [SBUK13] Sibay GE, Braberman V, Uchitel S, Kramer J (2013) Synthesizing modal transition systems from triggered scenarios. *IEEE Trans Softw Eng* 39(7):975–1001
- [SMMM98] Sutcliffe AG, Maiden NAM, Minocha S, Manuel D (1998) Supporting scenario-based requirements engineering. *IEEE Trans Softw Eng* 24(12):1072–1088
- [Som15] Somenzi F (2015) CUDD: CU decision diagram package release 3.0.0. <http://vlsi.colorado.edu/~fabio/CUDD/cudd.pdf>
- [Spe] Spectra Website. <http://smlab.cs.tau.ac.il/syntech/spectra/>
- [SUB08] Sibay G, Uchitel S, Braberman V (2008) Existential live sequence charts revisited. In: *Proceedings of the 30th international conference on software engineering*. ACM, pp 41–50
- [Tar55] Tarski A (1955) A lattice-theoretical fixpoint theorem and its applications. *Pac J Math* 5(2):285–309
- [UBC07] Uchitel S, Brunet G, Chechik M (2007) Behaviour model synthesis from properties and scenarios. In: *Proceedings of the 29th international conference on software engineering*. IEEE Computer Society, pp 34–43
- [UBC09] Uchitel S, Brunet G, Chechik M (2009) Synthesis of partial behavior models from properties and scenarios. *IEEE Trans Softw Eng* 35(3):384–406
- [VdHT10] Veanes M, de Halleux P, Tillmann N (2010) Rex: symbolic regular expression explorer. In: *Third international conference on software testing, verification and validation, ICST 2010, Paris, France, April 7–9, 2010*, pp 498–507
- [WR14] Walker A, Ryzhyk L (2014) Predicate abstraction for reactive synthesis. In: *Formal methods in computer-aided design, FMCAD 2014, Lausanne, Switzerland, October 21–24, 2014*. IEEE, pp 219–226
- [ZMT05] Zachos K, Maiden N, Tosar A (2005) Rich-media scenarios for discovering requirements. *IEEE Softw* 22(5):89–97

Received 27 February 2020

Accepted in revised form 30 January 2021 by Annabelle McIver, Maurice ter Beek and Cliff Jones

Published online 5 May 2021