GR(1)*: GR(1) Specifications Extended with Existential Guarantees

Gal Amram, Shahar Maoz, and Or Pistiner

Tel Aviv University

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 the 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, and induces only a minor additional cost in terms of time complexity, proportional to the extended length of the formula.

1 Introduction

Reactive synthesis is an automated procedure to obtain a correct-by-construction reactive system from its temporal logic specification [37]. 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 [36] (LTL), which has an efficient symbolic synthesis algorithm [6,35] and whose expressive power covers most of the well-known LTL specification patterns of Dwyer et al. [14,29]. GR(1) specifications include assumptions and guarantees about what needs to hold on all initial states, on all states (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 [24,28], scenario-based specifications [34], aspect languages [33], event-based behavior models [13], hybrid systems [17], and device drivers [47], to name a few.

A common form of providing system's requirements is through use cases [1,20,38]. 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 GR(1)*, which extends GR(1) specifications and synthesis with existential guarantees over input and output (environment and system) variables. GR(1)* allows engineers to naturally describe use cases as part of the specification and to efficiently synthesize a correct-by-construction controller that guaranties to make them possible.

We formally define GR(1)* and show that it is strictly more expressive than GR(1) (as it can express properties that are in CTL* [16] and outside LTL), see Sect. 3. We show how to solve GR(1)* games using a symbolic fixed-point algorithm, and present a corresponding controller construction, see Sect. 4 and Sect. 5. All proofs are provided in Appx. A.

Importantly, in comparison to GR(1) [6], GR(1)* induces only a minor additional cost in terms of time complexity, proportional to the extended length of the formula. Specifically, in [6] GR(1) games are solved in time $O(nmN^2)$ (measured in symbolic steps), 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 GR(1)* games is $O((nm + \ell r(k'))N^2)$, where l is the number of existential guarantees and $r(k')$ is the length of the longest existential guarantee.

The remainder of the introduction presents a running example and discusses related work.

1.1 Example: Lift Specification

As a motivating example, we enrich a lift specification, inspired by the example in [6], with several existential guarantees.

According to the original specification, the lift moves between n floors and must reach every floor it was called to. The environment controls button presses on every floor through variables $\{b_1, \ldots, b_n\}$. The system controls the lift's location through variables $\{f_1, \ldots, f_n\}$. The lift can move at most one floor in a single step, a button that has been pressed is turned off iff the lift reaches its floor, and the lift is required to reach every floor it was called to. For a complete description see [6]. We now describe example existential guarantees.

First, assume that the requirements document describes a typical use case: the lift is at floor i, button j is pressed, and the lift eventually reaches floor j. The engineer wants to integrate this use case into the specification in order to make sure that the lift will enable it. Thus, in our example, she formalizes the

following guarantees and adds them to the specification:

$A_{i,j}$ *Typical use case - the lift is at floor i, button j is pressed, and the lift eventually reaches floor j:* $GE(F(f_i \wedge b_j \wedge F(f_j)))$.

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. She thus formalizes and adds the following guarantees to the specification:

B_i *Button i can always be pressed:* $GE(F(b_i))$.

Finally, we present an example of using a negative scenario as a 'test'. Consider new system variables mUp and mDown that model the direction in which the lift should move. The engineer believes that in the presence of pending calls, the specified lift never stays in place. To test this hypothesis, she checks that the specification with the following additional existential guarantee is unrealizable:

C *The lift is in idle mode although there is a pending call:* $GE(F(\bigvee_{i=1}^{n} b_i \wedge \neg\mathsf{mUp} \wedge \neg\mathsf{mDown}))$.

1.2 Related Work

LTL synthesis of reactive systems was studied in [37] and shown to be 2EXPTIME-complete [39]. The GR(1) fragment of LTL, which can be solved in time quadratic in the size of the state space, is proposed in [6]. As GR(1)* augments GR(1) with existential requirements, it is in fact a fragment of CTL* [12,16]. Kupferman and Vardi showed that the synthesis problem for CTL* formulas is 2EXPTIME-complete [26]. Recently, Bloem et al. suggested a CTL* synthesis technique [7], 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. [19,25] studied synthesis of object systems from universal and existential live sequence charts (LSC). Uchitel et al. [40,41,45,46] studied synthesis of modal transition systems (MTS) from universal and existential message sequence charts (MSC). 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 [2,3,43,48]. All these motivated us to extend GR(1) with existential guarantees.

In the context of GR(1), Bloem et al. [5] 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. [15] and Majumdar et al. [27] 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 justice assumptions as existential guarantees, our technique allows to add any sequence of assertions as existential guarantees. Note that the problem solved in these papers is not a special case of our solution, since [15] and [27] 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.

2 Preliminaries

2.1 Game Structures and Strategies

Our notations are standard and mostly based on [6]. 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 $\mathcal{V}_1, \ldots, \mathcal{V}_k$ are pairwise disjoint sets of variables and $s_i \in 2^{\mathcal{V}_i}$, we write $(s_1, \ldots, s_k)$ as an abbreviation for $s_1 \cup \cdots \cup s_k$. Thus, $(s_1, \ldots, s_k)$ is a state over $\mathcal{V} = \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_k$.

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 a 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 $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}' \cup \mathcal{Y}'$. 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 $(s, s'_x) \models \rho^e$, but there is no $s_y \in 2^{\mathcal{Y}}$ such that $(s, s'_x, s'_y) \models \rho^s$. 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 $s_0, s_1, s_2, \ldots$, 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 $f^s : (2^{\mathcal{X} \cup \mathcal{Y}})^+ \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ such that (1) for $s_0 \in S$, (s_0) is *consistent* with f^s; (2) if $(s_0, \ldots, s_k)$ 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 $f^s(s_0, \ldots, s_k, s_x)$ is defined, and for $s_y = f^s(s_0, \ldots, s_k, s_x)$,

$(s_k, s'_x, s'_y) \models \rho^s$, and the sequence $(s_0, \ldots, s_k, (s_x, s_y))$ 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 $f^e : (2^{\mathcal{X}\cup\mathcal{Y}})^+ \to 2^{\mathcal{X}}$ 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 $(2^{\mathcal{X}\cup\mathcal{Y}})^+ \times 2^{\mathcal{X}}$. $C(s_0, \ldots, s_k, s_x) = (s_y, m)$ if, from state s_0, by receiving inputs $s_1|_{\mathcal{X}}, \ldots, s_k|_{\mathcal{X}}, s_x$, the controller replies with $s_1|_{\mathcal{Y}}, \ldots, s_k|_{\mathcal{Y}}, s_y$, 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 $(s_0, \ldots, s_k)$ is consistent with C where $s_0 \in S$, and for $s_x \in 2^{\mathcal{X}}$, $C(s_0, \ldots, s_k, s_x) = (s_y, m_{k+1})$. Then, $(s_0, \ldots, s_k, (s_x, s_y))$ 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 $C(s_0, \ldots, s_k, (s_x, s_y), t_x) = (t_y, m_{k+2})$.

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

2.2 Linear Temporal Logic and the GR(1) Fragment

Linear temporal logic (LTL) [36] is a language to specify properties over infinite words. Given a set of variables, $\mathcal{V}$, LTL formulas are generated by the grammar $\varphi = p|\neg\varphi|\varphi \vee \varphi|\varphi \wedge \varphi|X\varphi|F\varphi|G\varphi|\varphi U\varphi$, 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 in a standard way.

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. GS and φ 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. GS and φ. A strategy for the system, f^s, *wins* from $s \in 2^{\mathcal{X}\cup\mathcal{Y}}$ w.r.t. GS and φ if every play from s, consistent with f^s, wins for the system w.r.t. GS and φ. A strategy for the system, f^s is a *winning strategy* w.r.t. GS and φ 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. GS and φ. The *winning region* of the system includes all states, s, for which there exists a strategy for the system that wins from s. The winning region and a winning strategy for the environment are defined analogously.

Among LTL formulas, of special interest to us is the GR(1) fragment [6]. 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 $a_1, \ldots, a_m, g_1, \ldots, g_n$ are assertions. The assertions $a_1, \ldots, a_m$ are called *justice assumptions*, and $g_1, \ldots, g_n$ are called *justice guarantees*.

2.3 μ-calculus Over Game Structures

Modal μ-calculus [23] 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 [6], which uses the controllable predecessor operators $\langle\!\langle e \rangle\!\rangle$ and $[\![e]\!]$. In addition, to deal with possible behaviors, we add to the logic of [6] the predecessor operator $\Diamond$.

For a set of variables $\mathcal{V}$, and a set of relational variables $Var = \{X, Y, \ldots\}$, a μ-calculus (in positive form) formula is constructed by the grammar $\phi = p|X|\phi \vee \phi|\phi \wedge \phi|\langle\!\langle e \rangle\!\rangle\phi|[\![e]\!]\phi|\Diamond\phi|\mu X\phi|\nu X\phi$, where p is an assertion over $\mathcal{V}$. A μ-calculus formula defines a subset of $2^{\mathcal{V}}$. In words, $\langle\!\langle e \rangle\!\rangle\phi$ defines the set of states from which the system can enforce reaching next a state in the set that ϕ defines, $[\![e]\!]\phi$ defines the set of states from which the environment can enforce reaching next a state in the set that ϕ defines, and $\Diamond\phi$ is the set of states from which the environment and the system can choose an input and an output to reach 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 $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, a μ-calculus formula ϕ, and a valuation $\mathcal{E} : Var \rightarrow 2^{(2^{\mathcal{X}\cup\mathcal{Y}})}$, the semantics of ϕ, $[\![\phi]\!]^{\mathcal{E}}_{GS} \subseteq 2^{\mathcal{X}\cup\mathcal{Y}}$, is defined by a structural induction. We refer the reader to [6] for the exact definition, and add the rule: $[\![\Diamond\phi]\!]^{\mathcal{E}}_{GS} = \{s \in 2^{\mathcal{V}} : \exists t \in 2^{\mathcal{X}\cup\mathcal{Y}}(((s, t|'_{\mathcal{X}}, t|'_{\mathcal{Y}}) \models \rho^e \wedge \rho^s) \wedge t \in [\![\phi]\!]^{\mathcal{E}}_{GS})\}$.

If all relational variables in ϕ are bound by fixed-point operators, we omit the notation $\mathcal{E}$, and just write $[\![\phi]\!]_{GS}$. We remark that by Knaster-Tarski theorem [44], $[\![\mu X\phi]\!]^{\mathcal{E}}_{GS}$ and $[\![\nu X\phi]\!]^{\mathcal{E}}_{GS}$ indeed return the l.f.p. and g.f.p. of the function $S \mapsto [\![\phi]\!]^{\mathcal{E}[X\leftarrow S]}_{GS}$. This theorem can be applied since the positive form ensures that the function $S \mapsto [\![\phi]\!]^{\mathcal{E}[X\leftarrow S]}_{GS}$ is monotone, so l.f.p. and g.f.p. exist.

3 GR(1)*: Going Beyond LTL

3.1 GR(1)* Formulas

GR(1)* extends the GR(1) fragment of LTL with existential guarantees of the form $E(F(q_1 \wedge F(q_2 \wedge F(q_3 \wedge \cdots F(q_r) \cdots))))$ that should hold globally, where $q_1, \ldots, q_r$ are assertions over a set of variables $\mathcal{V}$. As in the case of justice guarantees, these guarantees should hold if all justice assumptions are satisfied infinitely often. Therefore, since this formula prescribes a possible behavior, the synthesized controller should either enable reaching $q_1, q_2, \ldots, q_r$ in that order, or enforce the violation of the assumptions.

Definition 1 (GR(1)*). *For $k \in \{1, \ldots, \ell\}$ let $S_k = E(F(q_{(k,1)} \wedge F(q_{(k,2)} \wedge F(q_{(k,3)} \wedge \cdots F(q_{(k,r(k))}) \cdots))))$. A GR(1)* formula over a set of variables $\mathcal{V}$ is*

a formula of the form

$$A\big(\bigwedge_{j=1}^{m} GF(a_j) \rightarrow (\bigwedge_{i=1}^{n} GF(g_i) \wedge \bigwedge_{k=1}^{\ell} G(S_k))\big),$$

where $a_1, \ldots, a_m, g_1, \ldots, g_n, q_{(1,1)}, \ldots, q_{(\ell,r(\ell))}$ *are assertions over* $\mathcal{V}$.

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.

Both syntactically and semantically, GR(1)* is a fragment of CTL*, and it is neither a subset of LTL nor of CTL, as illustrated in Fig. 1. For instance, the GR(1)* formula $A(GF(a) \rightarrow (GF(g) \wedge GE(F(q))))$ is expressible neither in LTL nor in CTL. This can be proved using arguments similar to [16].

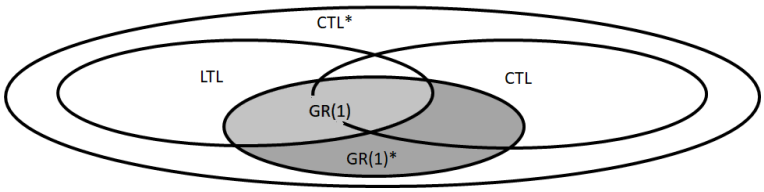

Fig. 1. Relationships between GR(1)* and other subsets of CTL*

3.2 GR(1)* Winning Condition

We now define when a strategy is winning w.r.t. a GR(1)* winning condition. As GR(1)* is a subset of CTL*, we essentially apply the general definition of winning strategies for reactive systems with CTL* winning conditions [26]. That is, roughly, we wish to say that a strategy is winning if it induces a computation tree which satisfies the GR(1)* formula. However, since, in contrast to [26], we consider game structures that restrict the steps that the players can perform, we cannot directly apply the definition of [26], 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 [26], and our formal definition takes this fact into consideration.

Definition 2. *Let* $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$ *be a game structure, and let* $\psi = A\big(\bigwedge_{j=1}^{m} GF(a_j) \rightarrow (\bigwedge_{i=1}^{n} GF(g_i) \wedge \bigwedge_{k=1}^{\ell} G(S_k))\big)$ *be a GR(1)* formula over* $\mathcal{X} \cup \mathcal{Y}$ *where* $S_k = E(F(q_{(k,1)} \wedge F(q_{(k,2)} \wedge F(q_{(k,3)} \wedge \cdots F(q_{(k,r(k))}) \cdots))))$. *A*

strategy for the system f^s wins from a state $s \in 2^{\mathcal{X} \cup \mathcal{Y}}$ w.r.t. GS and ψ, iff for every play π from s that is consistent with f^s, the following two requirements hold:

1. *If π is finite, then it ends in a deadlock for the environment.*
2. *If π is infinite and $\pi \models \bigwedge_{j=1}^{m} GF(a_j)$, then:*
   (a) *$\pi \models \bigwedge_{i=1}^{n} GF(g_i)$.*
   (b) *For every $i \geq 0$, and an existential guarantee, S_k, the ith prefix of π, $\pi(0), \ldots, \pi(i)$ can be extended to a play $\tilde{\pi}$, consistent with f^s, such that there are indices $i \leq i_1 \leq i_2 \leq \cdots \leq i_{r(k)}$ with $\tilde{\pi}(i_j) \models q_{(k,j)}$.*

Further, f^s is a winning startegy if for every $s_x \in 2^{\mathcal{X}}$ with $s_x \models \theta^e$, 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).

3.3 Inexpressibility of GR(1)* Winning Conditions in LTL

The fact that GR(1)* is a fragment of CTL* that is expressible neither in LTL nor in CTL (see, Sect. 3.1), 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 [16] 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. GS and the winning condition φ, iff it is winning w.r.t. GS and the winning condition ψ. Unsurprisingly, the example of [16] works in our context as well, and proves that GR(1)* winning conditions are inexpressible in LTL.

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

The proof of Prop. 1 reveals that even a winning condition as simple as $A(G(EF(y))$ is inexpressible in LTL. Using similar arguments one can show that every existential guarantee from our motivating example (Sect. 1.1) is inexpressible in LTL.

4 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 $A(\bigwedge_{j=1}^{m} GF(a_j) \rightarrow (\bigwedge_{i=1}^{n} GF(g_i) \wedge \bigwedge_{k=1}^{\ell} G(S_k)))$, where $S_k = E(F(q_{(k,1)} \wedge F(q_{(k,2)} \wedge F(q_{(k,3)} \wedge \cdots F(q_{(k,r(k))}) \cdots))))$. To compute the system's winning region, we present a μ-calculus formula that consists of three components: V, $f(Z)$, and $\{h_k(Z) \colon k = 1, \ldots, \ell\}$, 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 $\bigvee_{j=1}^{m} FG(\neg a_j)$. These states are characterized by the μ-calculus formula:

$$\mu Y(\bigvee_{j=1}^{m} \nu X(\bigcirc Y \vee (\neg a_j \wedge \bigcirc X))) \tag{1}$$

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

The second component we consider is the formula from [6] 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:

$$\nu Z(\bigwedge_{i=1}^{n} \mu Y(\bigvee_{j=1}^{m} \nu X((g_i \wedge \bigcirc Z) \vee \bigcirc Y \vee (\neg a_j \wedge \bigcirc X)))) = \nu Z(f(Z)) \tag{2}$$

For the third component, we turn to look at the existential guarantees of the GR(1)* formula, $\{S_k \colon 1 \leq k \leq \ell\}$. For every such guarantee S_k, and $1 \leq i \leq r(k) + 1$, we define a μ-calculus formula $h_{k,i}(Z)$, parametrized by a set of states Z. This formula characterizes the set of all states in Z from which there is a path in Z that traverses through $q_{(k,i)}, q_{(k,i+1)}, \ldots, q_{(k,r(k))}$. The formulas $h_{k,r(k)}, \ldots, h_{k,1}$ are defined recursively in reverse order as follows:

- $h_{k,r(k)+1}(Z) = Z$;
- $h_{k,i}(Z) = \mu Y((q_{(k,i)} \wedge h_{k,i+1}(Z)) \vee (Z \wedge \Diamond Y))$.

We are interested in the the outcome of this recursive formula.

$$h_{k,1}(Z) = h_k(Z) \tag{3}$$

The formula in Eq. 3 has k nested μ-operators. However, for $i > j$, the quantified variable of $h_{k,j}$ does not appear in $h_{k,i}$. Therefore, by computing the functions $h_{k,r(k)}(Z), \ldots, h_{k,1}(Z)$ in that order, $[\![h_k(Z)]\!]_{GS}$ is computed in $O(r(k) \cdot N)$ time.

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.

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

Theorem 1 (Realizabilty). *$[\![\nu Z(V \vee (f(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z)))]\!]_{GS}$ 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 $[\![\nu Z(V \vee (f(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z)))]\!]_{GS}$. Hence, to conclude the correctness of Thm. 1, we need to show that the system cannot win from every state in the complementary set of $[\![\nu Z(V \vee (f(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z)))]\!]_{GS}$.

Lemma 1. *If* $s \notin [\![\nu Z(V \vee (f(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z)))]\!]_{GS}$*, then no strategy for the system wins from* s*.*

By Thm. 1, a naive approach provides a realizability check for GR(1)* formulas in $O(nmN^3+lr(k')N^2)$ symbolic steps, where $r(k') = \max\{r(k)\colon 1 \leq k \leq \ell\}$. As in the case of GR(1), with the technique proposed in [8], this can be improved to $O((nm+lr(k'))N^2)$ steps. We see that extending GR(1) with existential guaranties adds only a minor cost to the realizability check.

5 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 very simple, and we start by describing it along with an informal description of a GR(1)* controller construction. Then, we point out that this construction has a significant disadvantage. We therefore suggest a way to improve the construction and present a detailed construction for our improved GR(1)* controller.

5.1 Construction Discussion and Overview

The basic idea is to alternate between two phases. Phase I is the well-known GR(1) strategy from [6]. The GR(1) controller satisfies the justice guarantees $g_1, \ldots, g_n$, 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 an existential guarantee S_k, and tries, cooperatively, to reach $q_{(k,1)}, \ldots, q_{(k,r(k))}$.

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 guaranties can be satisfied as well [5,15,27]. 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 some existential guarantee, S_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 $\bigcap_{k=1}^{\ell} [\![h_k(W)]\!]_{GS}$. Hence, we refer to $V' = W \setminus \bigcap_{k=1}^{\ell} [\![h_k(W)]\!]_{GS}$ 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 2. $V' \subseteq V$.

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. Whenever the game reaches V', we force violation of the justice assumptions.

5.2 Detailed Construction

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

$\mathbf{C_V}$. C_V is a memoryless controller that forces from V the violation of the assumptions so that it wins from V the game whose winning condition is $\bigvee_{j=1}^{m} FG(\neg a_j)$. Clearly, $\bigvee_{j=1}^{m} FG(\neg a_j) \equiv (\bigvee_{j=1}^{m} FG(\neg a_j)) \vee F(\mathsf{false} \wedge \bigcirc V)$. Hence, we apply the results of [21, Lem. 9] and construct C_V using the technique proposed in [6].

$\mathbf{C_{(GR(1),i)}}$. For each $1 \leq i \leq n$, $C_{(\mathrm{GR}(1),i)}$ is a memoryless controller that forces from $[\![f(W)]\!]_{GS}$ the satisfaction of the formula $(\bigvee_{j=1}^{m} FG(\neg a_j)) \vee F(g_i \wedge \bigcirc W)$. The construction of these controllers is described in [6].

$\mathbf{C_k}$. For $1 \leq k \leq \ell$, C_k is an $r(k)$-memory controller that, from each $s \in [\![h_k(W)]\!]_{GS}$, tries to walk on a path in W that traverses through $q_{(k,1)}, \ldots, q_{(k,r(k))}$. Possibly, for some states $(s, s_x) \in W \times 2^{\mathcal{X}}$, the controller has no defined response, and then the controller returns a designated value "undefined". We present the construction in Alg. 1. The controller C_k has a memory variable M, which stores values from $\{1, \ldots, r(k)\}$. M is initialized to 1.

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 [9] operations is straightforward.

In Alg. 2, we use a 4-field array M for the GR(1)* controller's memory. $M[0]$ specifies the next justice guarantee the controller strives to fulfil, and thus stores values from $\{1, \ldots, n\}$. $M[1]$ specifies the number of the ensuing existential guarantee we aim to satisfy, and thus stores values from $\{1, \ldots, \ell\}$. $M[2]$ is a field used by the controllers $C_1, \ldots, C_\ell$, which require memory of size equal to the number of states the controller is trying to traverse. Hence, $M[2]$ stores values from $\{1, \ldots, r(k')\}$ where $r(k') = \max\{r(k) \colon 1 \leq k \leq \ell\}$. Finally, $M[3]$ indicates which phase we now aim to. When $M[3] = 0$, we want to satisfy the justice guarantee $g_{M[0]}$ (phase I), and when $M[3] = 1$, we are trying, cooperatively, to satisfy the existential guarantee $S_{M[1]}$ (phase II). In line 15, $\oplus 1$ means that we write to $M[1]$ the next value. That is, for $k < \ell$, $k \oplus 1 = k+1$, and $\ell \oplus 1 = 1$. If the

Algorithm 1 The controller C_k

/* From state s */

input: $s_x \in 2^{\mathcal{X}}$ such that $(s, s'_x) \models \rho^e$

1: **if** $s \models q_{(k,M)} \wedge q_{(k,M+1)} \wedge \cdots \wedge q_{(k,r(k))}$ **then return** "undefined"
2: **end if**
3: $M \leftarrow \min\{M \leq t \leq r(k) \colon s \models \neg q_{k_t}\}$
4: **return** $C_{k,M}(s, s_x)$

/* The construction of $C_{k,t}$ for $t \in \{1, \ldots, r(k)\}$ */

1: $X \leftarrow \{s \in [\![h_k(W)]\!]_{GS} \colon s \models q_{(k,t)}\}$
2: **while** $X \neq [\![h_k(W)]\!]_{GS}$ **do**
3: $\quad X' \leftarrow \Diamond X \cap [\![h_k(W)]\!]_{GS}$
4: $\quad$ **for all** $s \in X' \setminus X$ and $s_x \in 2^{\mathcal{X}}$ such that $(s, s'_x) \models \rho^e$ **do**
5: $\quad\quad$ **if** there exists $s_y \in 2^{\mathcal{Y}}$ s.t. $(s_x, s_y) \in X$ and $(s, (s'_x, s'_y)) \models \rho^s$ **then**
6: $\quad\quad\quad$ choose $s_y \in 2^{\mathcal{Y}}$ s.t. $(s_x, s_y) \in X$ and $(s, (s'_x, s'_y)) \models \rho^s$
7: $\quad\quad\quad$ $C_{k,t}(s, s_x) \leftarrow s_y$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: $\quad X \leftarrow X \cup (\Diamond X \cap [\![h_k(W)]\!]_{GS})$
11: **end while**

play reaches a V'-state, the GR(1)* controller activates C_V for the remainder of the play.

The reader may notice three seemingly problematic issues concerning the construction that require clarification: (1) once the play reaches V', it must never leave, i.e., never reach $2^{\mathcal{X}\cup\mathcal{Y}} \setminus V'$; (2) the controllers $C_{(\mathrm{GR}(1),1)}, \ldots, C_{(\mathrm{GR}(1),n)}$ operate from $[\![f(W)]\!]_{GS}$, but the initialization in Alg. 2 returns a state in W; (3) phase I ends by reaching a state $s \models g_n \wedge \bigcirc\!\!\!\!\diamond W$, from which we want to activate some C_k, which operates from $[\![h_k(W)]\!]_{GS}$. The correctness proof we provide in the appendix settles these three 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) we show that $W \subseteq [\![f(W)]\!]_{GS}$;[1] (3) we show that $\bigcirc\!\!\!\!\diamond W \subseteq W$ and thus, if $s \models \bigcirc\!\!\!\!\diamond W$ then either $s \in V'$ and C_V can be activated, or $s \in \bigcap_{k=1}^{\ell} [\![h_k(W)]\!]_{GS}$ and each C_k can be activated.

Theorem 2 (Controller construction). *The controller C as constructed according to Alg. 2 implements a winning strategy for the system from W with memory of size $2 \cdot n \cdot \ell \cdot r(k')$.*

As a final remark, we note that in lines 5 and 7, the condition $(s_x, s_y) \models g_{M[0]} \wedge \bigcirc\!\!\!\!\diamond W$ can be replaced with $(s_x, s_y) \models g_{M[0]}$, for optimization purposes. We now explain the correctness of this replacement. In the proof of Thm. 2, we show that $\bigcirc\!\!\!\!\diamond W \subseteq W$. Moreover, Thm. 2 and Lem. 1 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

[1] In fact, they are equal, but we do not use this observation in the correctness proof of the algorithm.

Algorithm 2 The GR(1)* controller

/* Initialization */
input: $s_x \models \theta^e$
$M \leftarrow (1, 1, 1, 0)$
return s_y, such that $(s_x, s_y) \models (\theta^s \wedge W)$
/* From state s */
input: s_x such that $(s, s'_x) \models \rho^e$
if $s \models V'$ **then return** $C_V(s, s_x)$ // The play reached V'
end if
if $M[3] == 0$ **then** // Phase I: Fulfilling a justice guarantee
  $s_y \leftarrow C_{(\mathrm{GR}(1), M[0])}(s, s_x)$
  if $((s_x, s_y) \models g_{M[0]} \wedge \diamondsuit W) \wedge (M[0] < n)$ **then** $M[0] \leftarrow M[0] + 1$
  end if
  if $((s_x, s_y) \models g_{M[0]} \wedge \diamondsuit W) \wedge (M[0] = n)$ **then** $M[3] \leftarrow 1$
  end if
  return s_y
end if
if $M[3] == 1$ **then** // Phase II: Fulfilling an existential guarantee
  $(s_y, M[2]) \leftarrow C_{M[1]}(s, s_x, M[2])$
  if $s_y \neq$ "undefined" **then return** s_y
  end if
  $(M[0], M[1], M[2], M[3]) \leftarrow (1, M[1] \oplus 1, 1, 0)$
  goto line 1
end if

reachable by the GR(1)* controller, $s \models \diamondsuit W$ and there is no reason to test this condition.

6 Implementation and Preliminary Evaluation

We have implemented the realizability check for GR(1)* from Sect. 4 and integrated it in the Spectra language and synthesis environment [31,49], which already includes a GR(1) synthesizer and implementations of several additional analyses. Our implementation uses BDDs [9] via the CUDD 3.0 [42] package.

As a preliminary evaluation for the feasibility of GR(1)* in terms of running times, specifically the cost of adding existential guarantees to a GR(1) specification, we performed the following experiment.

6.1 Setup

We took the lift specification from our motivation example with 40 floors, 40 justice guarantees of the form $GF(\neg b_i)$, effectively meaning that in any infinite run, all requests will be served infinitely often, and 40 justice assumptions of the form $GF(b_i)$, meaning that in any infinite run, every floor will be requested infinitely often. We show the specification in Appx. B. This specification is well-separated [30] and realizable. Since the lift can only be positioned at a single

# ex. gar. \ length	2	5	10	15	20	25	30	35	40
2	178%	170%	164%	161%	153%	143%	142%	136%	138%
5	162%	154%	150%	139%	124%	124%	119%	142%	135%
10	175%	152%	156%	130%	141%	114%	124%	127%	134%
15	152%	149%	123%	116%	137%	125%	115%	119%	138%
20	137%	128%	125%	155%	142%	123%	142%	131%	137%
25	143%	135%	121%	123%	120%	134%	135%	114%	132%
30	155%	139%	144%	136%	134%	118%	124%	132%	144%
35	136%	137%	153%	133%	138%	148%	126%	128%	135%
40	138%	144%	145%	144%	111%	127%	139%	136%	128%

Table 1. Running times of 81 configurations with additional existential guarantees relative to the baseline specification of a lift with 40 floors, whose running time was 39178ms.

floor at a time, we model its location in Spectra by an integer typed variable, which is internally implemented as a $\lceil \log 40 \rceil$-bit variable (and not by a 40-bit array as in [6]). Hence, the size of our example's state space is 2^{46}.

Our goal is to evaluate the cost, in running time, of adding existential guarantees to a GR(1) specification. We thus measured realizability checking times of the baseline specification (without any existential guarantees) and compared it to realizability checking times of the same specification with number of existential guarantees ranging from 2 to 40, and length of existential guarantees (nesting depth of F) ranging from 2 to 40. We generated these existential guarantees by sampling a sequence of assertions of the required length, from some manually written list of assertions. As an example, Appx. B, l. 29 presents (in Spectra syntax) one of the existential guarantees we sampled: `GE floor = 17 & button[21], floor = 21`. In CTL*, this existential guarantee is written $GE(F(\texttt{floor} = 17 \wedge \texttt{button}[21] \wedge F(\texttt{floor} = 21)))$. The length of this CTL* formula, i.e., the number of nested F occurrences, is 2.

Overall, we considered 81 different configurations. For each configuration, we repeated realizability checking 15 times[2], and computed the median running time. We performed the experiments on hardware with Intel Xeon W-2133 processor with 32 GB RAM, running Windows 10.

6.2 Results

The median running time for the realizability check of the baseline was 39178 milliseconds. Table 1 shows the ratios between the running times for the different configurations and the baseline result (e.g., the leftmost cell in the table, reading

[2] Even though the algorithm is deterministic, we performed multiple runs since JVM garbage collection and the default automatic variable reordering of CUDD add variance to running times.

178%, means that with 2 existential guarantees of length 2, the median of 15 runs of realizability check took 78% longer than the baseline, i.e., about 70 seconds).

We observe that adding existential guarantees, as expected, has a cost. Yet, the growth in running time seems to be limited. In all 81 configurations, the running time was less than twice the running time of the baseline configuration. We further observe that neither increasing the number of existential guarantees nor extending their length significantly seems to affect running times.

The results of this preliminary evaluation are encouraging, but they are limited to a single specification and experiment setup. Further evaluation with additional specifications is required in order to strengthen the validity of the results.

7 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 enable them. We proved that GR(1)* captures CTL* properties that are inexpressible in LTL. We solved the realizability problem and presented a symbolic controller construction for GR(1)*. Importantly, GR(1)* realizability check consumes $O((nm + \ell r(k'))N^2)$-time where n is the number of justice guarantees, m is the number of justice assumptions, ℓ is the number of existential guarantees, and $r(k')$ is the length of the longest existential guarantee, i.e., induces only a minor additional cost in terms of time complexity, proportional to the extended length of the formula.

We consider the following future directions. First, GR(1) has been extended with past LTL operators (already in [6]) and patterns [29]. It would be valuable to carry over these extensions to GR(1)*, specifically supporting the use of past LTL operators and some of the patterns of [14] inside the existential guarantees.

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., [4,10,11,22,32]. It is interesting to investigate these in the context of GR(1)*. Specifically, counter-strategies for GR(1)* may be more complicated than counter-strategies for GR(1), as they depend not only on each play alone but also on the controller's possible behavior, i.e., a play can be winning for the system w.r.t. one controller, while losing w.r.t. another.

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).

References

1. Alexander, I.F., Maiden, N.: Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle. Wiley Publishing, 1st edn. (2004)

2. Alrajeh, D., Kramer, J., Russo, A., Uchitel, S.: Learning from vacuously satisfiable scenario-based specifications. In: International Conference on Fundamental Approaches to Software Engineering. pp. 377–393. Springer (2012)
3. Alrajeh, D., Ray, O., Russo, A., Uchitel, S.: Using abduction and induction for operational requirements elaboration. Journal of Applied Logic **7**(3), 275–288 (2009)
4. Alur, R., Moarref, S., Topcu, U.: 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. pp. 26–33. IEEE (2013), http://dx.doi.org/10.1109/FMCAD.2013.6679387
5. Bloem, R., Ehlers, R., Könighofer, R.: 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. Lecture Notes in Computer Science, vol. 9364, pp. 394–410. Springer (2015), https://doi.org/10.1007/978-3-319-24953-7
6. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of Reactive(1) Designs. J. Comput. Syst. Sci. **78**(3), 911–938 (2012), http://dx.doi.org/10.1016/j.jcss.2011.08.007
7. Bloem, R., Schewe, S., Khalimov, A.: CTL* synthesis via LTL synthesis. In: Proceedings Sixth Workshop on Synthesis, SYNT@CAV 2017, Heidelberg, Germany, 22nd July 2017. pp. 4–22 (2017), https://doi.org/10.4204/EPTCS.260.4
8. Browne, A., Clarke, E.M., Jha, S., Long, D.E., Marrero, W.R.: An improved algorithm for the evaluation of fixpoint expressions. Theor. Comput. Sci. **178**(1-2), 237–255 (1997), http://dx.doi.org/10.1016/S0304-3975(96)00228-9
9. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8), 677–691 (1986), https://doi.org/10.1109/TC.1986.1676819
10. Cavezza, D.G., Alrajeh, D.: Interpolation-based GR(1) assumptions refinement. In: TACAS. LNCS, vol. 10205, pp. 281–297 (2017), https://doi.org/10.1007/978-3-662-54577-5_16
11. Cimatti, A., Roveri, M., Schuppan, V., Tchaltsev, A.: Diagnostic information for realizability. In: VMCAI. LNCS, vol. 4905, pp. 52–67. Springer (2008), http://dx.doi.org/10.1007/978-3-540-78163-9_9
12. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Workshop on Logic of Programs. pp. 52–71. Springer (1981)
13. D'Ippolito, N., Braberman, V.A., Piterman, N., Uchitel, S.: Synthesizing nonanomalous event-based controllers for liveness goals. ACM Trans. Softw. Eng. Methodol. **22**(1), 9 (2013), http://doi.acm.org/10.1145/2430536.2430543
14. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE. pp. 411–420. ACM (1999)
15. Ehlers, R., Könighofer, R., Bloem, R.: 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. pp. 3478–3485. IEEE (2015), https://doi.org/10.1109/IROS.2015.7353862
16. Emerson, E.A., Halpern, J.Y.: "sometimes" and "not never" revisited: on branching versus linear time temporal logic. J. ACM **33**(1), 151–178 (1986), https://doi.org/10.1145/4904.4999
17. Filippidis, I., Dathathri, S., Livingston, S.C., Ozay, N., Murray, R.M.: 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. pp. 1030–1041. IEEE (2016), https://doi.org/10.1109/CCA.2016.7587949
18. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001], Lecture Notes in Computer Science, vol. 2500. Springer (2002), https://doi.org/10.1007/3-540-36387-4
19. Harel, D., Kugler, H.: Synthesizing state-based object systems from LSC specifications. International Journal of Foundations of Computer Science **13**(01), 5–51 (2002)
20. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (2004)
21. Kesten, Y., Piterman, N., Pnueli, A.: Bridging the gap between fair simulation and trace inclusion. Information and Computation **200**(1), 35 – 61 (2005), http://www.sciencedirect.com/science/article/pii/S0890540105000234
22. Könighofer, R., Hofferek, G., Bloem, R.: Debugging formal specifications: a practical approach using model-based diagnosis and counterstrategies. STTT **15**(5-6), 563–583 (2013), http://dx.doi.org/10.1007/s10009-011-0221-y
23. Kozen, D.: Results on the propositional μ-calculus. In: Proceedings of the 9th Colloquium on Automata, Languages and Programming. pp. 348–359. Springer-Verlag, London, UK, UK (1982), http://dl.acm.org/citation.cfm?id=646236.682866
24. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE Trans. Robotics **25**(6), 1370–1381 (2009), http://dx.doi.org/10.1109/TRO.2009.2030225
25. 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. pp. 445–460. Springer (2005)
26. Kupferman, O., Vardi, M.Y.: Synthesis with incomplete information. In: Advances in Temporal Logic, pp. 109–127. Springer (2000)
27. Majumdar, R., Piterman, N., Schmuck, A.: Environmentally-friendly GR(1) synthesis. CoRR **abs/1902.05629** (2019), http://arxiv.org/abs/1902.05629
28. Maniatopoulos, S., Schillinger, P., Pong, V., Conner, D.C., Kress-Gazit, H.: Reactive high-level behavior synthesis for an atlas humanoid robot. In: Kragic, D., Bicchi, A., Luca, A.D. (eds.) 2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016. pp. 4192–4199. IEEE (2016), https://doi.org/10.1109/ICRA.2016.7487613
29. Maoz, S., Ringert, J.O.: GR(1) synthesis for LTL specification patterns. In: ESEC/FSE. pp. 96–106. ACM (2015), http://doi.acm.org/10.1145/2786805.2786824
30. Maoz, S., Ringert, J.O.: 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. pp. 362–372. ACM (2016), http://doi.acm.org/10.1145/2950290.2950300
31. Maoz, S., Ringert, J.O.: Spectra: A specification language for reactive systems. CoRR **abs/1904.06668** (2019), http://arxiv.org/abs/1904.06668
32. Maoz, S., Ringert, J.O., Shalom, R.: Symbolic repairs for GR(1) specifications. In: Mussbacher, G., Atlee, J.M., Bultan, T. (eds.) Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada,

May 25-31, 2019. pp. 1016–1026. IEEE / ACM (2019), https://dl.acm.org/citation.cfm?id=3339632
33. Maoz, S., Sa'ar, Y.: AspectLTL: an aspect language for LTL specifications. In: AOSD. pp. 19–30. ACM (2011), http://doi.acm.org/10.1145/1960275.1960280
34. Maoz, S., Sa'ar, Y.: Assume-guarantee scenarios: Semantics and synthesis. In: MODELS. LNCS, vol. 7590, pp. 335–351. Springer (2012), http://dx.doi.org/10.1007/978-3-642-33666-9_22
35. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: VMCAI. LNCS, vol. 3855, pp. 364–380. Springer (2006), http://dx.doi.org/10.1007/11609773_24
36. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977), https://doi.org/10.1109/SFCS.1977.32
37. Pnueli, A., Rosner, R.: On the Synthesis of a Reactive Module. In: POPL. pp. 179–190. ACM Press (1989)
38. Pohl, K.: Requirements Engineering: Fundamentals, Principles, and Techniques. Springer Publishing Company, Incorporated, 1st edn. (2010)
39. Rosner, R.: Modular synthesis of reactive systems. Ph.D. thesis, Weizmann Institute of Science (1992)
40. Sibay, G., Uchitel, S., Braberman, V.: Existential live sequence charts revisited. In: Proceedings of the 30th international conference on Software engineering. pp. 41–50. ACM (2008)
41. Sibay, G.E., Braberman, V., Uchitel, S., Kramer, J.: Synthesizing modal transition systems from triggered scenarios. IEEE Transactions on Software Engineering **39**(7), 975–1001 (2013)
42. Somenzi, F.: CUDD: CU Decision Diagram Package Release 3.0.0. http://vlsi.colorado.edu/~fabio/CUDD/cudd.pdf (2015)
43. Sutcliffe, A.G., Maiden, N.A., Minocha, S., Manuel, D.: Supporting scenario-based requirements engineering. IEEE Transactions on software engineering **24**(12), 1072–1088 (1998)
44. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics **5**(2), 285–309 (1955), https://doi.org/10.2140/pjm.1955.5.285
45. Uchitel, S., Brunet, G., Chechik, M.: Behaviour model synthesis from properties and scenarios. In: Proceedings of the 29th international conference on Software Engineering. pp. 34–43. IEEE Computer Society (2007)
46. Uchitel, S., Brunet, G., Chechik, M.: Synthesis of partial behavior models from properties and scenarios. IEEE Transactions on Software Engineering **35**(3), 384–406 (2009)
47. Walker, A., Ryzhyk, L.: Predicate abstraction for reactive synthesis. In: Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014. pp. 219–226. IEEE (2014), http://dx.doi.org/10.1109/FMCAD.2014.6987617
48. Zachos, K., Maiden, N., Tosar, A.: Rich-media scenarios for discovering requirements. IEEE software **22**(5), 89–97 (2005)
49. Spectra Website. http://smlab.cs.tau.ac.il/syntech/spectra/

A Proofs

Proof (Proof of Prop. 1). Take a variable y, and consider the formula $A(G(EF(y))$. This is a GR(1)* formula as it is equivalent to $A(GF(\texttt{true}) \rightarrow G(EF(y)))$. Towards a contradiction, assume that the winning condition $A(G(EF(y))$ is equivalent to an LTL winning condition φ. Consider the following game structure,

$$GS = \langle \mathcal{X} = \{x\}, \mathcal{Y} = \{y\}, \theta^e = \neg x, \theta^s = \neg y, \rho^e = \texttt{true}, \rho^s = \texttt{true} \rangle.$$

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, in each step, f^s assigns **true** to variable y iff the environment assigns **true** to x. It is easy to see that f^s is a winning strategy w.r.t. the winning condition $A(G(EF(y))$.

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, and so $\pi \models \varphi$. Hence, f^s is also winning in the second game structure w.r.t. φ. On the other hand, in the second game structure, f^s is clearly not winning w.r.t. $A(G(EF(y))$, which proves that the winning condition φ is not equivalent to $A(G(EF(y)))$. □

Proof (Proof of Lem. 1).

Recall the seminal result that ω-regular games (and thus LTL games) are determined [18], i.e., for $GS = (\mathcal{X}, \mathcal{Y}, \theta^e, \theta^s, \rho^e, \rho^s)$, and an LTL formula over $\mathcal{X} \cup \mathcal{Y}$, φ, the winning regions of the system and the environment w.r.t. GS and φ form a partition of $2^{\mathcal{X} \cup \mathcal{Y}}$. We use this fact below.

$[\![\nu Z(V \vee (f(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z)))]\!]_{GS} = \bigcap_{t=0}^{\infty} Z_t$ where $Z_0 = 2^{\mathcal{X} \cup \mathcal{Y}}$ and $Z_{t+1} = [\![V \vee (f(Z_t) \wedge \bigwedge_{k=1}^{\ell} h_k(Z_t))]\!]_{GS}$. We show, by induction, that the claim holds for each $s \in 2^{\mathcal{X} \cup \mathcal{Y}} \setminus Z_t$. This suffices since $Z_0 \supseteq Z_1 \supseteq \ldots$ and thus, since the state space is finite, $[\![\nu Z(V \vee (f(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z)))]\!]_{GS} = Z_r$ for a 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, for which the condition in Def. 2 does not hold. Since $Z_{t+1} = [\![V \vee (f(Z_t) \wedge \bigwedge_{k=1}^{\ell} h_k(Z_t))]\!]_{GS}$, $s \notin V$ and either $s \notin [\![\mu Y(\bigvee_{j=1}^{m} \nu X((g_i \wedge \bigcirc Z_t) \vee \bigcirc Y \vee (\neg a_j \wedge \bigcirc X)))]\!]_{GS}$ for some i, or $s \notin [\![h_k(Z_t)]\!]_{GS}$ for some k.

First, if $s \notin [\![\mu Y(\bigvee_{j=1}^{m} \nu X((g_i \wedge \bigcirc Z_t) \vee \bigcirc Y \vee (\neg a_j \wedge \bigcirc X)))]\!]_{GS}$, the claim follows from the correctness of Eq. 2. Specifically, by [21, Lem. 9], the system cannot win from s w.r.t. the winning condition $(\bigvee_{j=1}^{m} FG(\neg a_j)) \vee F(g_i \wedge \bigcirc Z_t)$. 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: $\bigwedge_{j=1}^{m} GF(a_j) \wedge G(\neg g_i \vee \bigcirc\!\!\!\!\bigcirc \neg Z_t)$. 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 $\hat{\pi}$ for which the condition in Def. 2 does not hold.

Now, in case that $s \notin [\![h_k(Z_t)]\!]_{GS}$, 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 traverses through $q_{(k,1)}, \ldots, q_{(k,r(k))}$. We distinguish between two possibilities. First, suppose that every play 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, $\pi(0), \ldots, \pi(i)$, of π. By assumption, every extension of this prefix to a play $\hat{\pi}$, consistent with f^s, never leaves Z_t. Therefore, the suffix of such play $\hat{\pi}^i$, does not traverses through $q_{(k,1)}, \ldots, q_{(k,r(k))}$ and hence, the existential guarantee S_k does not hold from $\pi(0), \ldots, \pi(i)$, which proves that the condition in Def. 2 does not hold for π.

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 $\pi(0), \ldots, \pi(i)$ can be extended into a play $\hat{\pi}$, consistent with f^s, for which the condition in Def. 2 does not hold. □

Proof (Proof of Lem. 2). Take $s \in V'$, and assume, towards a contradiction, that $s \notin V$. Since $W = V \cup ([\![f(W)]\!] \cap \bigcap_{k=1}^{\ell} [\![h_k(W)]\!]_{GS})$, $s \in \bigcap_{k=1}^{\ell} [\![h_k(W)]\!]_{GS}$, in contradiction to $s \in V'$. □

Proof (Proof of Thm. 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. Alg. 3 is obtained from Alg. 2 by the addition of line 12. To prove the correctness of Alg. 3, first, we show that the construction indeed defines a properly working controller, and then we show that it wins the GR(1)* game.

We need to show that from any reachable state and memory value, for every legal input, the controller replies with a legal output. Clearly, it suffices to show that the following hold:

1. If $s \in V'$ and $(s, s'_x) \models \rho^e$, then $(s_x, C_V(s, s_x)) \in V'$.
2. When the play traverses to phase I, its current state $s \in [\![f(W)]\!]_{GS}$.
3. When the play traverses to phase II, its current state $s \in \Diamond W$, and if $s \in W$, then $s \in \bigcap_{k=1}^{n} [\![h_k(W)]\!]_{GS}$.

Item 1 is proved as follows: Write $s_y = C_V(s, s_x)$ and assume, towards a contradiction, that $(s_x, s_y) \notin V'$. But, $(s_x, s_y) \in V$ thus $(s_x, s_y) \in W$. Therefore, for every existential guarantee S_k, $(s_x, s_y) \in [\![h_k(W)]\!]_{GS}$. Clearly, since $(s, (s'_x, s'_y)) \models \rho^e \wedge \rho^s$, this implies that also $s \in [\![h_k(W)]\!]_{GS}$ and hence, $s \notin V'$ in contradiction to the statement's assumption.

To prove the second item, we prove the following claim.

Claim 1. $W \cup \Diamond W \subseteq [\![f(W)]\!]_{GS}$.

Algorithm 3 A modification of the GR(1)* controller construction

/* Initialization */

input: $s_x \models \theta^e$

1: $M \leftarrow (1, 1, 1, 0)$

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** $s \models V'$ **then return** $C_V(s, s_x)$ // The play reached V'

2: **end if**

3: **if** $M[3] == 0$ **then** // Phase I: Fulfilling a justice guarantee

4: $s_y \leftarrow C_{(\mathrm{GR}(1), M[0])}(s, s_x)$

5: **if** $((s_x, s_y) \models g_{M[0]} \wedge \bigcirc\!\!\!\!\diamond W) \wedge (M[0] < n)$ **then** $M[0] \leftarrow M[0] + 1$

6: **end if**

7: **if** $((s_x, s_y) \models g_{M[0]} \wedge \bigcirc\!\!\!\!\diamond W) \wedge (M[0] = n)$ **then** $M[3] \leftarrow 1$

8: **end if**

9: **return** s_y

10: **end if**

11: **if** $M[3] == 1$ **then** // Phase II: Fulfilling an existential guarantee

12: **if** $s \in \bigcirc\!\!\!\!\diamond W \setminus W$ **then return** s_y s.t. $(s_x, s_y) \in W$ // new line

13: **end if**

14: $(s_y, M[2]) \leftarrow C_{M[1]}(s, s_x, M[2])$

15: **if** $s_y \neq$ "undefined" **then return** s_y

16: **end if**

17: $(M[0], M[1], M[2], M[3]) \leftarrow (1, M[1] \oplus 1, 1, 0)$

18: **goto** line 1

19: **end if**

Proof (Claim 1.). First, we show that $V \subseteq [\![f(W)]\!]_{GS}$. For each justice guarantee, g_i, let $U_i = [\![\mu Y(\bigvee_{j=1}^{m} \nu X((g_i \wedge \bigcirc\!\!\!\!\diamond W) \vee \bigcirc\!\!\!\!\diamond Y \vee (\neg a_j \wedge \bigcirc\!\!\!\!\diamond X)))]\!]_{GS}$. Hence, $[\![f(W)]\!]_{GS} = \bigcap_{i=1}^{n} U_i$. By [21, Lem. 9], U_i is the system's winning region for the game whose winning condition is $(\bigvee_{j=1}^{m} FG(\neg a_j)) \vee F(g_i \wedge \bigcirc\!\!\!\!\diamond W)$. Hence, clearly, $V \subseteq U_i$ for each $1 \leq i \leq n$ thus $V \subseteq \bigcap_{i=1}^{n} U_i = [\![f(W)]\!]_{GS}$.

Now, W is a fixed-point of the formula $V \vee (f(Z) \wedge \bigwedge_{k=1}^{\ell} h_k(Z))$. Namely, $W = [\![V \vee (f(W) \wedge \bigwedge_{k=1}^{\ell} h_k(W))]\!]_{GS} = V \cup ([\![f(W)]\!]_{GS} \cap \bigcap_{k=1}^{\ell} [\![h_k(W)]\!]_{GS})$. Since $V \subseteq [\![f(W)]\!]_{GS}$, $W = [\![f(W)]\!]_{GS} \cap \bigcap_{k=1}^{\ell} [\![h_k(W) \vee V]\!]_{GS}$ thus $W \subseteq [\![f(W)]\!]_{GS}$. Moreover, it follows that $W \subseteq U_i$ for each $1 \leq i \leq n$. Therefore, by definition of U_i, also $\bigcirc\!\!\!\!\diamond W \subseteq U_i$ and we get that $\bigcirc\!\!\!\!\diamond W \subseteq [\![f(W)]\!]_{GS}$. $\square_{\text{Claim 1}}$

Now we can prove that item 2 holds. Consider a play $s_0, s_1, \ldots$, consistent with the GR(1)* controller in Alg. 3, and assume that the game traverses to phase I in the step (s_i, s_{i+1}). Hence, either $i = 0$, or s_i is the final state induced by a sub-play consistent with phase II. If $i = 0$, by the initialization, $s_i \in W$ and then, by the former claim, $s_i \in [\![f(W)]\!]_{GS}$. Now, if s_i is a final state reached in phase II, then, again, $s_i \in W$. This holds since every play consistent with each of the controllers $C_1, \ldots, C_\ell$ never reaches out of W. Hence, the former claim handles this case as well.

We now turn to prove that item 3 holds. Consider a play $s_0, s_1, \ldots$, consistent with the GR(1)* controller in Alg. 3, and assume that the game traverses to phase II in the step (s_i, s_{i+1}). Hence, s_i is the final state reached in phase I, and hence, $s_i \models g_n \wedge \bigcirc\!\!\!\!\diamond W$. To complete the proof for this item, assume that $s \in W$. Since the play traversed to phase II, the condition in line 1 does not hold, and $s \notin V'$. Therefore, $s \in W \setminus V' = \bigcap_{k=1}^{n} [\![h_k(W)]\!]_{GS}$, as required.

Now, we turn to prove that the GR(1)* controller, presented in Alg. 3, wins w.r.t. the GR(1)* winning condition. Consider a play, π, consistent with that controller. If the assumptions are violated in π, we are done. Otherwise, the play never activates C_V, and all justice guarantees are satisfied infinitely often.

Consider a state $s = \pi(i)$, and an existential guarantee S_k. We need to show that the prefix $\pi(0), \ldots, \pi(i)$, can be completed into a play $\hat{\pi}$, consistent with the GR(1)* controller, such that its suffix $\hat{\pi}(i+1), \hat{\pi}(i+2), \ldots$ traverses through $q_{(k,1)}, \ldots, q_{(k,r(k))}$.

In the suffix of π that starts from s, since all justice guarantees are satisfied infinitely often, at some point $M[2]$ stores k and $M[3]$ stores 1. Namely, there is a state $t = \pi(j)$, such that $j > i$ and from which C_k is activated until it returns: "undefined". Therefore, there exists another play, $\hat{\pi}$, consistent with the GR(1)* controller, that starts with the prefix $\pi(0), \ldots, \pi(j) = t$, and its suffix $\hat{\pi}(j+1), \ldots$, traverses through $q_{(k,1)}, \ldots, q_{(k,r(k))}$, 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 $\bigcirc\!\!\!\!\diamond W$: 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. 1, the system has no winning strategy from any state $s \in 2^{\mathcal{X} \cup \mathcal{Y}} \setminus W$. Hence, $\bigcirc\!\!\!\!\diamond W \subseteq W$. Consequently, line 12 in Alg. 3 deals with a vacuous situation and it is redundant. Clearly, the correctness of Alg. 2 follows. □

B Lift Specification

```
spec Lift40402
// Boolean array through which the environment notifies of
    requests on every floor
env boolean[40] button;
// System integer variable that controls the lift's position
sys Int(0..39) floor;
// Initially, there are no requests
asm ini forall i in Int(0..39). !button[i];
// Initially, the lift should be at the first floor
gar ini floor=0;
// The lift is allowed to move at most one floor at a time
gar G next(floor) = floor+1 | next(floor)=floor | next(floor) =
    floor-1;
// Once a request has been fulfilled, it is removed
asm G forall i in Int(0..39). (button[i] & floor=i) -> next(!
    button[i]);
// Requests cannot be withdrawn
asm G forall i in Int(0..39). (button[i] & floor!=i) -> next(
    button[i]);

// 40 justice guarantees: eventually satisfy every request
gar GF !button[0];
    ...
gar GF !button[39];

// 40 justice assumptions: each floor is requested infinitely
    often
asm GF button[0];
    ...
asm GF button[39];

// 40 existential guarantees of length 2: The lift is at floor i
     when floor j is requested; eventually, floor j is reached
gar scenario0: // GE(F(floor=17 & button[21] & F(floor=21)))
GE floor = 17 & button[21], floor = 21;
    ...
gar scenario39: // GE(F(floor=6 & button[4] & F(floor=4)))
GE floor = 6 & button[4], floor = 4;
```

Listing 1. An example Spectra [31,49] specification for a lift with 40 floors, augmented with existential guarantees. The specification consists of environment variables **env**, system variables **sys**, environment assumptions **asm**, and system guarantees **gar**.