Robustness in the Presence of Liveness*

Roderick Bloem[1], Krishnendu Chatterjee[2], Karin Greimel[1], Thomas A. Henzinger[2], and Barbara Jobstmann[3]

[1] Graz University of Technology
[2] IST Austria (Institute of Science and Technology Austria)
[3] CNRS/Verimag

Abstract. Systems ought to behave reasonably even in circumstances that are not anticipated in their specifications. We propose a definition of robustness for liveness specifications which prescribes, for any number of environment assumptions that are violated, a minimal number of system guarantees that must still be fulfilled. This notion of robustness can be formulated and realized using a Generalized Reactivity formula. We present an algorithm for synthesizing robust systems from such formulas. For the important special case of Generalized Reactivity formulas of rank 1, our algorithm improves the complexity of [PPS06] for large specifications with a small number of assumptions and guarantees.

1 Introduction

Current verification and synthesis approaches consider the functional correctness of a system as a Boolean question: either the specification is fulfilled, or it is not. This approach is unsatisfactory in many situations [BCHJ09]. In particular, many specifications consist of environment assumptions and system guarantees. For such specifications, the classical approach does not impose any restrictions on the behavior of the system when the environment assumptions are *not* fulfilled. We argue that (1) desirable systems act in some "reasonable" way, even if the environment does not always fulfill the assumptions and (2) it is an undue burden on the user to specify the proper behavior of the system for each and every environment behavior. Desirable systems should fulfill a natural "graceful degradation" property in the sense that the system should fulfill the guarantees as well as it can, given any behavior of the environment.

We have previously studied the verification and synthesis of robust systems for safety specifications [BGHJ09]. In the case of safety, environment failures are immediately apparent and the difficulty is how the system can best recover from them. A violation of a liveness property, however, cannot be detected at any point in time [AS85]. Thus, a system that is robust to liveness failures must attempt to fulfill its guarantees under all circumstances, without knowing whether the environment satisfies the assumptions.

* This work was supported by EU grants 217069 (COCONUT), 248613 (DIAMOND), 215543 (COMBEST), and the European Network of Excellence ArtistDesign.

T. Touili, B. Cook, and P. Jackson (Eds.): CAV 2010, LNCS 6174, pp. 410–424, 2010.
© Springer-Verlag Berlin Heidelberg 2010

In this paper, we define several possible notions of robustness in the presence of liveness, all aiming at maximizing the set of guarantees that is fulfilled for any set of fulfilled assumptions. Suppose a specification has two assumptions and two guarantees. In order for the specification to hold, both guarantees must be met when both assumptions are. A system that meets both guarantees when only one assumption is met is more robust than one that meets one (or zero) guarantees when one assumption is met.

Example 1. We consider a variant of the dining philosophers problem [Dij68]. There are n philosophers sitting at a round table. There is one chopstick between each pair of adjacent philosophers. Because each philosopher needs two chopsticks to eat, adjacent philosophers cannot eat simultaneously. We are interested in schedulers that use input variables h_i signifying that philosopher i is hungry and output variables e_i signifying that philosopher i is eating.

We have the following requirements. First, an eating philosopher prevents her neighbors from eating. Formally, $G_{1i} = \Box(e_i \rightarrow \neg e_{(i-1)\mathrm{mod}n} \wedge \neg e_{(i+1)\mathrm{mod}n})$. Second, an eating philosopher eats until she is no longer hungry: $G_{2i} = \Box(e_i \wedge h_i \rightarrow \bigcirc e_i)$. Third, every hungry philosopher eats eventually $G_{3i} = \Box(h_i \rightarrow \Diamond e_i)$. We add the assumption that an eating philosopher eventually loses her appetite: $A_{1i} = \Box(e_i \rightarrow \Diamond \neg h_i)$. Our final specification consists of n assumptions and $3n$ guarantees: $\bigwedge_{i=1}^{n} A_{1i} \rightarrow \bigwedge_{i=1}^{n}(G_{1i} \wedge G_{2i} \wedge G_{3i})$.

We have synthesized a system realizing this specification for 5 philosophers using our synthesis tool RATSY[1]. The system constructed by RATSY is not very robust: When philosopher 1 violates the assumption by always being hungry, then philosophers 1 and 3 eat forever, while the other philosophers starve. Thus the three guarantees $\Box(h_2 \rightarrow \Diamond e_2)$, $\Box(h_4 \rightarrow \Diamond e_4)$, and $\Box(h_5 \rightarrow \Diamond e_5)$ are violated. A more robust system would let philosopher 3 and 4 take turns, thus violating only two guarantees. □

In this paper, we consider Generalized Reactivity specifications of rank 1 (GR(1) specifications). GR(1) is an expressive specification formalism with a natural distinction between assumptions and guarantees [PPS06]. Efficient tools exist for GR(1) specifications, which have been used to synthesize relatively large specifications [JGWB07, BGJ+07]. GR(1) specifications are of the form $\varphi \rightarrow \psi$. Here, φ represents the environment assumptions and ψ represents the system guarantees and both φ and ψ are given as a set of deterministic Büchi automata. These automata are combined into a product automaton with state space Q, transition relation δ, and acceptance condition $\bigwedge_{i=1}^{m} \Box \Diamond a_i \rightarrow \bigwedge_{i=1}^{n} \Box \Diamond g_i$.

GR(1) specifications do not require any guarantees to be fulfilled when some assumption is violated. We propose an intuitive notion of robustness that prescribes, for any number of environment assumptions that is violated, a minimal number of system guarantees that must still be fulfilled. We show that this and related measures of robustness can be transformed to a specification of the form $\bigwedge_{j=1}^{k}(\bigwedge_{i=1}^{m} \Box \Diamond a_{ji} \rightarrow \bigwedge_{i=1}^{n} \Box \Diamond g_{ji})$, which is a Generalized Reactivity (generalized Streett) formula of rank k. We address the problem of verification

[1] `http://rat.fbk.eu/ratsy/index.php/Main/HomePage`

and especially of synthesis of such formulas, which allows us to construct robust systems.

The verification problem is a relatively straightforward generalization of the verification problem for GR(1) (cf. [GBJV08]) and can be performed in time $O(m \cdot n \cdot |Q| \cdot |\delta|)$. Recall that m is the number of assumptions and n is the number of guarantees, and $|Q|$ and $|\delta|$ refer respectively to the size of the state space and the transition relation of the product automaton.

The synthesis question is answered by solving a Generalized Reactivity game. This can either be done through a specialization of Zielonka's algorithm, or through a novel algorithm presented in this paper, both of which can be implemented symbolically. Zielonka's algorithm runs in time $O(|Q|^{2 \cdot k} \cdot |\delta| \cdot (m+n)^k \cdot k!)$, which we improve to $O(|Q|^k \cdot |\delta| \cdot (m \cdot n)^{k \cdot (k+1)} \cdot k!)$. On the other hand, our algorithm produces larger strategies and thus larger robust systems: the systems produced by Zielonka's algorithm have size $|Q| \cdot n^k \cdot k!$, whereas our algorithm produces systems of size $|Q| \cdot ((m+1) \cdot (n+1))^k \cdot k!$.

Our algorithm is a generalization of a game-theoretic algorithm for the important class of GR(1) conditions based on a reduction (via a counting construction) to Streett games with single pair. The algorithm runs in time $O(|Q| \cdot |\delta| \cdot (m \cdot n)^2)$. This bound improves the $O(|Q|^2 \cdot |\delta| \cdot m \cdot n)$ time bound of the algorithm of [PPS06] for the case that Q is larger than m and n, which is typical in such applications as GR(1) synthesis.

Measures of robustness for different fault models, for example internal malfunctions of circuits [FD08], have been studied. Classical notions of fault tolerance such as self-stabilization [Dij74] and the notions of closure and convergence suggested in [Aro93] focus on safety properties. Convergence requires that a system restores its invariant after an error has occurred, and closure requires that the system satisfies a second, larger invariant even when errors recur. Our approach can be viewed as an extension of closure to liveness, where we require that some weaker set of guarantees is fulfilled when the environment behaves unexpectedly. Apart from our previous work [BGHJ09], there is little work on synthesis of robust systems, although people have studied the related problem of retrofitting fault tolerance to existing programs. (See, e.g., [KE05, EKA08].)

The flow of the paper is as follows. After giving the necessary notation in Section 2, we define several notions of robustness in Section 3. In order to solve the synthesis problem for robust systems, we introduce the necessary transformations on the formulas and game theoretic algorithms in Sections 4 and 5. In Section 6 we return to the questions of verification and synthesis of robust systems. We conclude with Section 7.

2 Preliminaries

We consider systems with a set of input signals I and a set of output signals O. We define $AP = I \cup O$. We use the signals as atomic propositions in the specifications defined below. Our input alphabet is thus $\Sigma_I = 2^I$, the output alphabet is $\Sigma_O = 2^O$, and we define $\Sigma = 2^{AP}$.

Acceptance Conditions. The specifications we use are automata and we synthesize a system that realizes a given specification using games. Both automata and games can have the following acceptance conditions. Let Q be a set of states, an *acceptance condition* is a predicate $\mathrm{Acc} : Q^\omega \to \mathbb{B}$, mapping infinite runs to true or false (accepting and not accepting, or winning and losing, respectively). The *Büchi acceptance condition* is $\mathrm{Acc}(\rho) = 1$ iff $\inf(\rho) \cap F \neq \emptyset$, where $F \subseteq Q$ is the set of accepting states and $\inf(\rho)$ is the set of elements that occur infinitely often in ρ. We abbreviate the Büchi condition as $\mathcal{B}(F)$. A *Generalized Reactivity acceptance condition* is a predicate $\bigwedge_{l=1}^{k}(\bigwedge_{i=1}^{m_l} \mathcal{B}(A_{l,i}) \to \bigwedge_{i=1}^{n_l} \mathcal{B}(G_{l,i}))$, where $A_{l,i} \subseteq Q$ are assumptions and $G_{l,i} \subseteq Q$ are guarantees. To simplify notation, we will assume that the m_l are all equal to some constant m, and similarly for n_l and n. The acceptance condition is a *GR(1) acceptance condition* if $k = 1$, it is a *generalized Büchi acceptance condition* if $k = 1$ and $m = 0$, it is a *Streett acceptance condition* with k pairs if $m = n = 1$.

Automata. A *(complete deterministic) automaton* A over the alphabet Σ is a tuple $A = (Q, q_0, \delta, \mathrm{Acc})$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition function, and Acc is the acceptance condition. A *run* of an automaton A on a word $w = w_0 w_1 \ldots \in \Sigma^\omega$ is the sequence $\rho(w) = \rho_0 \rho_1 \ldots \in Q^\omega$ such that $\rho_0 = q_0$, and $\rho_{i+1} = \delta(\rho_i, w_i)$. An automaton accepts a word if its run is accepting ($\mathrm{Acc}(\rho(w)) = 1$); its language $L(A)$ consists of the set of words it accepts. A *Büchi automaton* $A = (Q, q_0, \delta, F)$ is an automaton with a Büchi condition with accepting state set F.

Generalized Reactivity(1) specifications. Consist of two parts: *assumptions* and *guarantees* [PPS06]. They specify the interaction between an environment (controlling the input variables Σ_I) and a system (controlling the output variables Σ_O). The specification states that the system must fulfill all guarantees whenever the environment fulfills all assumptions.

A GR(1) specification over the alphabet Σ consists of m Büchi automata $A_1^a, \ldots, A_m^a$ for the environment assumptions and n Büchi automata $A_1^g, \ldots, A_n^g$ for the system guarantees. [PPS06]. Let $A^{GR(1)} = (Q, \delta, q_0, \mathrm{Acc})$ be the product of all automata A_i^a and A_i^g, where the state space is $Q = Q_1^a \times \cdots \times Q_m^a \times Q_1^g \times \cdots \times Q_n^g$, the transition function is $\delta((q_1^a, \ldots, q_n^g), \sigma) = (\delta_1^a(q_1^a, \sigma), \ldots, \delta_n^g(q_n^g, \sigma))$, and the initial state is $q_0 = (q_{0,1}^a, \ldots, q_{0,n}^g)$. Let $J_i^a = \{(q_1^a, \ldots, q_n^g) \in Q \mid q_i^a \in F_i^a\}$ be the set of states that are accepting in A_i^a. Similarly, let J_i^g be the set of all states of $A^{GR(1)}$ that are accepting in A_i^g. The acceptance condition Acc is a GR(1) condition with assumptions J_i^a and guarantees J_i^g.

Note that the size of the state space of the specification grows exponentially with the number of assumptions and guarantees (if the Büchi automata have more than 2 states), whereas m and n grow linearly.

A system realizes a GR(1) specification $A^{GR(1)}$ if the language of the system is part of the language of $A^{GR(1)}$.

Games and Strategies. A *game graph* is a finite directed graph $G = (S, s_0, E)$ consisting of a set of states S, an initial state $s_0 \in S$, and a set of edges $E \subseteq S \times S$ such that each state has at least one outgoing edge. The states are partitioned

into a set S_1 of *Player-1 states* and a set S_2 of *Player-2 states*. When the initial state is not relevant, we omit it and write (S, E). A *play* $\rho = s_0 s_1 \ldots \in S^\omega$ is an infinite sequence of states such that for all $i \geq 0$ we have $(s_i, s_{i+1}) \in E$. Given a game graph $G = (S, E)$, a *(finite memory) strategy* for Player 1 is a tuple (Γ, γ_0, π), where Γ is some (finite) set representing the memory, $\gamma_0 \in \Gamma$ is the initial memory content, and $\pi : S_1 \times \Gamma \to S \times \Gamma$ is a function mapping a Player-1 state s and a memory content to a successor state s' and an updated memory content such that $(s, s') \in E$. A Player-2 strategy is defined similarly. A strategy is *positional* if it depends only on the current state. We represent a positional strategy π for player p as a function from S_p to S. Let $\rho((\Gamma_1, \gamma_{0,1}, \pi_1), (\Gamma_2, \gamma_{0,2}, \pi_2), s)$ denote the unique play starting at s when Player 1 plays according to the strategy $(\Gamma_1, \gamma_{0,1}, \pi_1)$ and Player 2 plays according to $(\Gamma_2, \gamma_{0,2}, \pi_2)$.

A *game* is a tuple $((S, E), \mathrm{Acc})$, consisting of a game graph (S, E) and an objective Acc. The game graph defines the possible actions of the players. The objective describes the winning condition for the players. A play ρ is winning for Player 1 if it satisfies the objective of the game, otherwise it is winning for Player 2. A strategy π_1 is winning for Player 1 if for all strategies π_2 of Player 2 the play $\rho((\Gamma_1, \gamma_{0,1}, \pi_1), (\Gamma_2, \gamma_{0,2}, \pi_2), s_0)$ is winning. A game is winning for Player 1 (Player 2) if there exists a winning strategy for Player 1 (Player 2, resp.). A *Generalized Reactivity (GR) game* is a game with a Generalized Reactivity acceptance condition, and similarly for GR(1).

Given a game graph G, two objectives are equivalent if all plays in G have the same winner for both objectives. The objectives are *equivalent* if they are equivalent for any game graph.

GR(1) Synthesis. A GR(1) specification can easily be translated into a GR(1) game. A winning strategy for the GR(1) game corresponds to a system that realizes the GR(1) specification.

3 Defining Measures of Robustness

In this section we discuss how to compare systems with respect to robustness. Usually, multiple systems satisfy a specification, but which one is most robust? In prior work we answered this question for safety specifications: our measure of robustness for a safety specification $\varphi \to \psi$ is the ratio between how often the environment violates φ and how often the system violates ψ. For specifications with liveness properties, this approach does not work because we cannot count the number of violations of a liveness property. Instead, we propose to count the number of properties violated. In the following we show two different robustness measures, the single and the multiple counting requirements measure. Then we formally state the requirements a robustness measure has to satisfy.

Single Counting Requirements. Recall the dining philosophers example with $n = 5$ philosophers given in the introduction. Suppose system D_1 always lets one philosopher eat until she is not hungry anymore and then moves to the next hungry philosopher in a round robin manner. If one philosopher is hungry

forever, then no other philosopher gets to eat again. Thus, the violation of one assumption leads to the violation of four guarantees.

Suppose system D_2 lets two non-adjacent philosophers eat at the same time until neither is hungry anymore. They take turns in the following order: first philosopher 1 and 3 eat, then philosopher 2 and 4, and last philosopher 3 and 5 eat. If one of the currently eating philosopher is hungry forever, then the two currently eating philosophers eat forever and no other philosopher gets to eat again. Thus, the violation of one assumption leads to the violation of three guarantees. System D_2 is thus more robust than system D_1.

An even more robust system (D_3) is the one described in the introduction. Two philosophers eat at the same time, as soon as one of them is not hungry anymore another philosopher with free chopsticks is allowed to eat. If one philosopher is hungry forever, she eats forever and the other philosophers that are not her neighbors take turns eating. The violation of one assumption leads to the violation of two guarantees.

We specify robust systems by adding restrictions to the original specification. All three systems above satisfy the original specification $\varphi = \bigwedge_{i=1}^{n} A_{1i} \rightarrow \bigwedge_{i=1}^{n}(G_{1i} \wedge G_{2i} \wedge G_{3i})$, but only D_2 and D_3 guarantee that they violate at most three system guarantees if the environment violates one of its assumptions. Formally, D_2 and D_3 additionally satisfy

$$\psi_1 = \Big(\bigvee_{i=1}^{n} \bigwedge_{j\in\{1,\ldots,n\}\setminus\{i\}} A_{1j}\Big) \rightarrow \Big(\varphi_S \wedge \bigvee_{i=1}^{n} \bigvee_{j=i+1}^{n} \bigvee_{k=j+1}^{n} \bigwedge_{l\in\{1,\ldots,n\}\setminus\{i,j,k\}} G_{3l}\Big),$$

where $\varphi_S = \bigwedge_{i=1}^{n}(G_{1i} \wedge G_{2i})$. The antecedent of the formula states that the environment satisfies $n-1$ out of the n assumptions. The consequent says that the system satisfies all the safety guarantees (G_{1i} and G_{2i}) but might violate three of its liveness guarantees.

Note that in general, a robust system cannot violate a safety guarantee in response to a violation of a fairness assumption, since a violation of a fairness assumption can not be detected in finite time.

Since D_3 violates at most two system guarantees if one environment assumption is violated, it also satisfies the following formula.

$$\psi_2 = \Big(\bigvee_{i=1}^{n} \bigwedge_{j\in\{1,\ldots,n\}\setminus\{i\}} A_{1j}\Big) \rightarrow \Big(\varphi_S \wedge \bigvee_{i=1}^{n} \bigvee_{j=i+1}^{n} \bigwedge_{k\in\{1,\ldots,n\}\setminus\{i,j\}} G_{3k}\Big)$$

These two formulas allow us to distinguish between systems D_1, D_2, and D_3, which satisfy the same base specification but differ in how resilient they are with respect to violated environment assumptions. We propose to use formulas of this type, which relate the number of satisfied assumptions to a number of satisfied guarantees to measure how robust a system is.

Suppose $\mathcal{A}$ is a set of assumptions and $\mathcal{G}$ is a set of guarantees. Let $\mathcal{A}_k = \{A \subseteq \mathcal{A} \mid |A| = k\}$ be the set of all subsets of $\mathcal{A}$ of size k and let $\mathcal{G}_k$ be defined similarly. We can augment the specification with a restriction of the form $(\bigvee_{A\in\mathcal{A}_k} \bigwedge_{A_i\in A} A_i) \rightarrow (\bigvee_{G\in G_l} \bigwedge_{G_i\in G} G_i)$ to check if a system satisfies l

guarantees when k assumptions are satisfied. Naturally, a system that satisfies more guarantees with the same number of satisfied assumptions is more robust.

Multiple Counting Requirements. In some cases we might want to have a more fine-grained measure of robustness, which cannot be expressed by a single restriction of the form given above. Recall again the dining philosophers example but this time assume there are $n = 7$ philosophers. Suppose system D_4 allows two hungry philosophers to eat at the same time. Then, even if one philosopher does not stop eating, the other non-adjacent philosophers can still take turns eating. However, if two philosophers misbehave and they both get to eat (i.e., they do not sit next to each other), they will leave the other five philosophers to starve. Suppose another system D_5 allows three philosophers to eat at the same time. Now, if two philosophers misbehave and they both get to eat, the system D_5 still allows another philosopher to eat and only four philosophers are left to starve. Both D_4 and D_5 realize the specification φ. If we consider the restrictions from above, we see that both systems satisfy the formula ψ_1 and ψ_2. Our previous measure of robustness cannot distinguish between D_4 and D_5. Let's add another restriction ψ_3 to our specification:

$$\psi_3 = \Big(\bigvee_{i=1}^{n} \bigvee_{j=i+1}^{n} \bigwedge_{k\in\{1,\ldots,n\}\setminus\{i,j\}} A_{1k}\Big) \rightarrow \Big(\varphi_S \wedge \bigvee_{i=1}^{n} \bigvee_{j=i+1}^{n} \bigvee_{k=j+1}^{n} \bigwedge_{l\in\{1,\ldots,n\}\setminus\{i,j,k\}} G_{3l}\Big)$$

System D_5 realizes $\varphi \wedge \psi_2 \wedge \psi_3$ but system D_4 does not. We can measure the number of satisfied guarantees for several numbers of satisfied assumptions. The restrictions we add to the specifications are of the form $\bigwedge_{(k,l)\in L}((\bigvee_{\mathcal{A}\in\mathcal{A}_k} \bigwedge_{A_i\in\mathcal{A}} A_i) \rightarrow (\bigvee_{\mathcal{G}\in\mathcal{G}_l} \bigwedge_{G_i\in\mathcal{G}} G_i))$, where L is a list of pairs (k, l), requiring l guarantees to be satisfied if k assumptions are satisfied.

Definitions. Both single and multiple counting requirements, as defined above, can be put in the following form (as we will shown in Section 4).

Definition 1. *Given a GR(1) specification $A^{GR(1)}$ with assumptions $J_1^a, \ldots, J_m^a$ and guarantees $J_1^g, \ldots, J_n^g$, a* robustness specification *for $A^{GR(1)}$ has the form*

$$\bigwedge_{l=1}^{k} \left(\bigwedge_{i=1}^{m_l} \mathcal{B}(J_{l,i}^a) \rightarrow \bigwedge_{i=1}^{n_l} \mathcal{B}(J_{l,i}^g) \right),$$

where $J_{l,i}^a \in \{J_1^a, \ldots, J_m^a\}$ and $J_{l,i}^g \in \{J_1^g, \ldots, J_n^g\}$.

There is a natural partial order on robustness specifications: If, for each set of satisfied assumptions, a specification S requires a superset of the guarantees required by specification S', then S is more robust than S'. Let us denote this order by $\prec$.

Definition 2. *A* robustness measure *for a GR(1) specification is a set of robustness specifications together with a total order that respects $\prec$.*

For example, consider again the 'simple counting requirements' robustness specifications from above. A possible total order is $(k = 0, l = |\mathcal{G}|) > (k = 0, l = |\mathcal{G}| - 1) > \ldots > (k = 0, l = 1) > (k = 1, l = |\mathcal{G}|) > \ldots > (k = |\mathcal{A}|, l = 0)$, where k is the number of satisfied assumptions and l the number of satisfied guarantees. Another possible total order is $(k = 0, l = |\mathcal{G}|) > (k = 1, l = |\mathcal{G}|) > \ldots > (k = |\mathcal{A}| - 1, l = |\mathcal{G}|) > (k = 0, l = |\mathcal{G}| - 1) > \ldots > (k = |\mathcal{A}|, l = 0)$. A total order is necessary to synthesize the most robust specification.

Section 6 shows how to verify and synthesize robust systems for a given measure. To synthesize a robust system, we solve games with the robustness specification as objective. Section 5 shows how to solve such games. In the next section, we show how to translate combinations of Büchi objectives to generalized Büchi objectives.

4 Simplification of Combinations of Büchi Objectives

In this section we present a simplification of disjunctions of conjunctions of Büchi objectives (DCB objectives) to conjunctions of Büchi objectives (generalized Büchi objectives). This simplification is needed to transform counting requirements to robustness specifications. The simplification (or reduction) incurs an exponential blowup. Games with generalized Büchi objectives can be solved in polynomial time, whereas we show that games with DCB objectives are coNP-complete. This shows that the exponential blow up in the translation is probably inevitable.

Simplification of DCB objectives. The simplification is done in two steps. First, we show how to translate DCB objectives to conjunctions of disjunctions of Büchi objectives. Second, we show that conjunctions of disjunctions of Büchi objectives can be translated to generalized Büchi objectives.

Lemma 1. *Any winning condition ψ that is a DCB objective can be translated into an equivalent winning condition ψ' that is a conjunction of disjunctions of Büchi objectives, such that $|\psi'| = O(2^{|\psi|})$.*

Proof. For any objective $\psi = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n} \mathcal{B}(B_{ij})$ there exists an equivalent objective $\psi' = \bigwedge_{i=1}^{n^m} \bigvee_{j=1}^{m} \mathcal{B}(B'_{ij})$ with $B'_{ij} \in \{B_{ij} \mid i \in \{1 \ldots m\} \text{ and } j \in \{1 \ldots n\}\}$. The translation is identical to that of changing DNF into CNF. □

Lemma 2. *Any winning condition ψ that is a conjunction of disjunctions of Büchi objectives can be translated into an equivalent generalized Büchi objective ψ', such that $|\psi'| = O(|\psi|)$.*

Proof. Since a disjunction of Büchi conditions is again a Büchi condition ($\mathcal{B}(B_1) \vee \mathcal{B}(B_2) = \mathcal{B}(B_1 \cup B_2)$), objectives of the form $\bigwedge_{i=1}^{k} \bigvee_{j=1}^{l} \mathcal{B}(B_{ij})$ can be reduced to a generalized Büchi objective $\bigwedge_{i=1}^{k} \mathcal{B}(\bigcup_{j=1}^{l} B_{ij})$. □

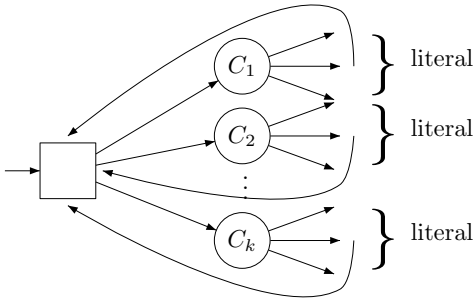

Fig. 1. Game graph for 3SAT formula

Corollary 1. *Any winning condition ψ that is a DCB objective can be translated into an equivalent generalized Büchi objective ψ', such that $|\psi'| = O(2^{|\psi|})$.*

Complexity of solving DCB objectives. We first show that the problem of deciding whether Player 1 has a winning strategy for a DCB objective is coNP-hard, and then we will argue coNP-completeness.

Hardness proof. We show that the problem of deciding whether Player 1 has a winning strategy in a game with a DCB objective is at least as hard as deciding whether a 3SAT formula is unsatisfiable. Consider a 3SAT formula ψ in CNF with clauses $C_1, C_2, \ldots, C_k$ over variables $\{x_1, x_2, \ldots, x_n\}$, where each clause consists of disjunctions of exactly three literals (a literal is a variable or its complement). Given the formula ψ, we construct a game graph as shown in Figure 1. The game graph is as follows: from the initial state, Player 1 chooses a clause, then from a clause Player 2 chooses a literal that appears in the clause (i.e., makes the clause true). From every literal the next state is the initial state. The winning condition for Player 1 is $\bigvee_{i=1}^{n}(\mathcal{B}(X_i) \wedge \mathcal{B}(\overline{X_i}))$, where X_i is the set of states that correspond to the literal x_i and $\overline{X_i}$ is the set of states that correspond to the complement literal $\neg x_i$; in other words, Player 1 wants to visit some variable and its complement infinitely often.

We now present two directions of the hardness proof.

Not satisfiable implies winning. We show that if ψ is not satisfiable, then Player 1 has a winning strategy. The winning strategy is as follows: the strategy is played in rounds; in round i Player 1 chooses the clauses $C_1, C_2, \ldots, C_k$ in order, and then proceeds to round $i + 1$. Since ψ is not satisfiable, for every round i there is at least one variable such that both the variable state and its complement state is visited in round i. Since the number of variables is finite, it follows that there must be some variable such that both the variable state and its complement state appears infinitely often. The result follows.

Satisfiable implies not winning. We now show that if ψ is satisfiable, then Player 2 has a winning strategy. Consider a satisfying assignment to ψ. A memoryless winning strategy for Player 2 is as follows: for every clause C_i, Player 2 chooses a literal from C_i that is set true by the satisfying assignment. Given the

strategy of Player 2, since the strategy is obtained from a valid assignment, it follows that never a variable and its complement is visited.

The above argument gives us the following lemma.

Lemma 3. *Given a game graph with a DCB objective, deciding if Player 1 has a winning strategy is coNP-hard.*

Lemma 4. *Given a game graph with a DCB objective, deciding if Player 1 has a winning strategy can be achieved in coNP.*

Proof. The proof is as follows: we have already shown that DCB objectives can be translated to a generalized Büchi objective (which is an upward-closed objective). It follows from the result of Zielonka [Zie98] that there are memoryless winning strategies for the complement of an upward-closed objective (in particular for disjunction of coBüchi objectives). It follows that there always exist memoryless winning strategies for Player 2. Hence to falsify that Player 1 has a winning strategy, a memoryless strategy for Player 2 can be fixed (as the polynomial witness) and the resulting one-player graph can be verified in polynomial time. The polynomial time verification procedure uses the following fact: consider a maximal strongly connected component (MSCC) in a one-player graph (only Player 1), then the MSCC is winning if for some index i of the disjunction, for every index j of the corresponding conjunction the MSCC contains at least one Büchi state B_{ij}. Using the above fact, MSCC decomposition of a graph, and reachability to winning MSCCs we obtain a polynomial time verification procedure. The result follows. □

Lemma 3 and Lemma 4 yield the following result.

Theorem 1. *Given a game graph with a DCB objective for Player 1, deciding if Player 1 has winning strategy is co-NP complete.*

5 Solving Generalized Reactivity Games

In this section, we first present a translation of GR(1) winning conditions to one-pair Streett conditions (or parity $\{0, 1, 2\}$ conditions). Our reduction is based on a counting construction similar to the reduction of generalized Büchi conditions to Büchi conditions. Second, we generalize the translation to reduce games with Generalized Reactivity objectives to games with Streett objectives.

Reduction. Consider a GR(1) game $G = ((S, E), \mathrm{Acc})$ with $\mathrm{Acc} = \bigwedge_{i=1}^{m} \mathcal{B}(A_i) \rightarrow \bigwedge_{i=1}^{n} \mathcal{B}(G_i)$ with Player 1 states S_1 and Player 2 states S_2. We construct an equivalent one-pair Streett game $G' = ((S', E'), \mathcal{B}(A'_1) \rightarrow \mathcal{B}(G'_1))$ with Player 1 states S'_1 and Player 2 states S'_2 as follows.

1. The state space $S' = S \times \{0, 1, \ldots, m\} \times \{0, 1, \ldots, n\}$, with $S'_1 = S_1 \times \{0, 1, \ldots, m\} \times \{0, 1, \ldots, n\}$, and $S'_2 = S_2 \times \{0, 1, \ldots, m\} \times \{0, 1, \ldots, n\}$.
2. The set of edges E' is defined as follows:

$((s,i,n),(s,0,0)) \in E'$ for $0 \leq i \leq m$,
$((s,m,j),(s,0,j)) \in E'$ if $j \neq n$, and
$((s,i,j),(s',i',j')) \in E'$ if $(s,s') \in E, i'=i+1$ if $s' \in A_{i+1}$ otherwise $i'=i$, and $j'=j+1$ if $s' \in G_{j+1}$ otherwise $j'=j$.

3. The Streett pair is $(A'_1 = \{(s,m,j) \in S' \mid j \in \{0,\ldots,n\}\}, G'_1 = \{(s,i,n) \in S' \mid i \in \{0,\ldots,m\}\})$.

We present the intuition behind the construction. Initially i and j are zero. If all the assumptions are visited such that, assumption A_2 is visited after some visit to assumption A_1; assumption A_3 is visited after some visits to assumptions A_1, A_2; assumption A_4 is visited after some visits to assumptions A_1, A_2, A_3; and so on, since the last reset, then i is reset to 0. If all the guarantees are visited, such that guarantee G_2 is visited after some visit to guarantee G_1; guarantee G_3 is visited after some visits to guarantees G_1, G_2; guarantee G_4 is visited after some visits to guarantees G_1, G_2, G_3; and so on, since the last reset, then j is reset to 0. In between resets, i and j denote the last assumption and the last guarantee visited in the order described above, since the last reset. The size of the new state space is $|S'| = |S| \cdot (m+1) \cdot (n+1) = O(|S| \cdot m \cdot n)$. The new number of transitions is $|E'| = |E| \cdot (m+1) \cdot (n+1) + 2 \cdot |S| = O(|E| \cdot m \cdot n)$.

Lemma 5. *There exists a winning strategy for G iff there exists a winning strategy for G'.*

Proof. Consider a play ρ in G and the corresponding play ρ' in G'. We consider two cases. Case one. We consider the case where all guarantees appear infinitely often in ρ. If all guarantees are visited infinitely often, then a state with third state component with value n is visited infinitely often in ρ' (i.e., G'_1 is visited infinitely often). Thus, if the play in G satisfies the GR(1) condition by visiting all guarantees infinitely often, then the corresponding play in G' visits G'_1 infinitely often and satisfies the Streett condition.

Case two. We consider the case where some guarantee is not visited infinitely often in ρ. In this case a state with third state component with value n is visited only finitely often in ρ'. We consider two sub-cases.

Case two(a). If all the assumptions are visited infinitely often in ρ, then a state with second state component with value m is visited infinitely often in ρ'. In this case the play in G does not satisfy the GR(1) condition, and the corresponding play in G' visits A'_1 infinitely often and G'_1 finitely often, which violates the Streett condition.

Case two(b). If some assumption is not visited infinitely often in ρ, then a state with second state component with value m is visited only finitely often in ρ' (i.e., A'_1 is visited finitely often). In this case the play in G satisfies the GR(1) condition, and the corresponding play in G' satisfies the Streett condition. This completes the proof. □

Theorem 2. *Games with GR(1) objectives can be solved in $O(|S| \cdot |E| \cdot (m \cdot n)^2)$ time.*

Since one-pair Streett (or parity $\{0, 1, 2\}$)) games with $|S|$ states and $|E|$ edges can be solved in $O(|S| \cdot |E|)$ time [Jur00], from Lemma 5 we obtain the above theorem. It may also be noted that one-pair Streett games can be solved very efficiently in practice [dAF07] and also symbolically [EJ91] (and implementing our counting construction symbolically is standard). The previous best know algorithm to solve GR(1) games was through the triple nested fix-point algorithm of [PPS06] which works in time $O(|S|^2 \cdot |E| \cdot n \cdot m)$. For the typical case that $|S|$ is much greater than m and n, our algorithm is faster.

Our algorithm can easily be generalized to Generalized Reactivity objectives.

Theorem 3. *Games with Generalized Reactivity objectives can be solved in* $O(|S|^k \cdot |E| \cdot (m \cdot n)^{k \cdot (k+1)} \cdot k!)$ *time.*

Proof. Turn all GR(1) objectives into Streett pairs, the Streett game has $O(|S| \cdot m^k \cdot n^k)$ states, $O(|E| \cdot m^k \cdot n^k)$ transitions, and k-Streett pairs. A Streett game with k pairs, $|E'|$ transitions and $|S'|$ states can be solved in $O(|E'| \cdot |S'|^k \cdot k!)$ [PP06]. □

A symbolic algorithm for Generalized Reactivity objectives can be obtained as follows: use the standard symbolic implementation of the counting construction along with the symbolic algorithm for Streett games from [PP06]. This gives us a symbolic algorithm for solving games with Generalized Reactivity objectives.

Winning strategy and memory required. A winning strategy for a GR(k) condition is obtained as follows: first we consider an automaton A_1 of size $((n+1) \cdot (m+1))^k$ to store the values of the counters and follow the transition as given in the reduction to Streett games with k pairs (essentially this mimics the reduction of the counting construction). Winning strategies in Streett games with k pairs require at most $k!$ memory, and a winning strategy (automata A_2 with $k!$ memory) can be constructed from the Streett game solving algorithms (such as [PP06] or [CHP07]). The product automaton $A_1 \times A_2$ describes a winning strategy for the GR(k) condition and requires $((n+1) \cdot (m+1))^k \cdot k!$ memory.

In the case of GR(1) conditions, our construction of winning strategies requires $(n+1) \cdot (m+1)$ memory. The memory can be improved to n as follows: once the winning set is computed, we can run Zielonka's algorithm to compute a winning strategy with n memory. However, as the winning set is already computed we can get rid of the outer iteration of Zielonka's algorithm and re-running Zielonka's algorithm to compute the winning strategy, given the winning set, takes $O(|S| \cdot |E| \cdot (n+m))$ time.

6 Verification and Synthesis of Robust Systems

First, we show how to verify whether a system has a certain level of robustness. Then, we give an algorithm to synthesize the most robust system with respect to a given robustness measure.

Verification. Verification of a robustness specification is similar to the verification of a GR(1) specification.

Lemma 6. *Given a GR(1) specification* $A^{GR(1)} = (Q, \delta, q_0, \mathrm{Acc})$ *with* m *assumptions and* n *guarantees, and a system* M*, verification can be performed in* $O(m \cdot n \cdot |Q|^2 \cdot |\delta|)$ *time.*

Proof. Check if a trace in $A^{GR(1)} \times M$ satisfies $\bigwedge_{i=1}^{m} \mathcal{B}(A_i) \wedge (\bigvee_{i=1}^{n} \neg \mathcal{B}(G_i))$ (the negation of the specification) using the μ-calculus formula $\mu X.(pre(X) \vee \bigvee_{j=1}^{n} \nu Y.(\neg G_j \wedge \bigwedge_{i=1}^{m} pre(\mu Z.(Y \wedge (A_i \vee pre(Z))))))$ [GBJV08]. The complexity of the nested fix-points is in $O(|Q|^2 \cdot |\delta|)$ [EL86]. □

Theorem 4. *Given a GR(1) specification* $A^{GR(1)} = (Q, \delta, q_0, \mathrm{Acc})$*, a robustness specification* $\bigwedge_{l=1}^{k}(\bigwedge_{i=1}^{m} \mathcal{B}(A_{l,i}) \rightarrow \bigwedge_{i=1}^{n} \mathcal{B}(G_{l,i}))$*, and a system* M*, verifying that* M *satisfies the robustness specification takes* $O(k \cdot m \cdot n \cdot |Q|^2 \cdot |\delta|)$ *time.*

Proof. Check if a trace in $A^{GR(1)} \times M$ satisfies the negation of the specification $\bigvee_{l=1}^{k}(\bigwedge_{i=1}^{m} \mathcal{B}(A_{l,i}) \wedge (\bigvee_{i=1}^{n} \neg \mathcal{B}(G_{l,j})))$ by checking the k different GR(1) parts $(\bigwedge_{i=1}^{m} \mathcal{B}(A_{l,i}) \wedge (\bigvee_{i=1}^{n} \neg \mathcal{B}(G_{l,j})))$ separately, one after the other, using the method of Lemma 6. □

Synthesis. The most robust system with respect to a given robustness measure can be synthesized by synthesizing the greatest realizable robustness specification. Thus, synthesis can be reduced to solving GR games.

Theorem 5. *Given a GR(1) specification* $A^{GR(1)} = (Q, \delta, q_0, \mathrm{Acc})$*, and a robustness measure with* h *robustness specifications* $r_p = \bigwedge_{l=1}^{k}(\bigwedge_{i=1}^{m} \mathcal{B}(A_{l,i}) \rightarrow \bigwedge_{i=1}^{n} \mathcal{B}(G_{l,i}))$*, with* $1 \leq p \leq h$*, and a total order, synthesis of the most robust system can be performed in* $O(h \cdot |Q|^k \cdot |\delta| \cdot (m \cdot n)^{k \cdot (k+1)} \cdot k!)$ *time. The size of the resulting system is* $((m+1) \cdot (n+1))^k \cdot k! \cdot |Q|$*.*

Proof. The best system can be synthesized by trying the specifications in order. Start with the lagest robustness specification according to the given total order. Try to synthesize a system satisfying the specification using the algorithm given in Section 5. The translation of the specification into a game graph is linear, hence synthesis of a robustness specification can be performed in $O(|Q|^k \cdot |\delta| \cdot (m \cdot n)^{k \cdot (k+1)} \cdot k!)$ time (see Theorem 3). The size of the synthesized system is $((m+1) \cdot (n+1))^k \cdot k! \cdot |Q|$, if the robustness specification is realizable. If the robustness specification is not realizable proceed with the next specification in the given order. □

7 Conclusions

We have presented a framework for robustness for liveness specifications. The notion of robustness that we suggest aims to maximize the number of guarantees that are fulfilled for any number of assumptions that may be violated. We have discussed several different interpretations of this notion and have shown that they can all be reduced to Generalized Reactivity formulas. We have shown how to verify such formulas and how to synthesize them to robust systems.

For synthesis we have developed a novel game-theoretic algorithm that is faster than Zielonka's, although it does produce strategies with larger memory. Our algorithm can also be used for the synthesis of GR(1) properties, in which case it outperforms the algorithm of [PPS06] when the state space of the specification is larger than the number of assumptions and guarantees.

References

[Aro93] Arora, A.: Closure and convergence: A foundation of fault-tolerant computing. IEEE Transatcions of Software Engineering 19, 1015–1027 (1993)

[AS85] Alpern, B., Schneider, F.B.: Defining liveness. Information Processing Letters 21, 181–185 (1985)

[BCHJ09] Bloem, R., Chatterjee, K., Henzinger, T., Jobstmann, B.: Better quality in synthesis through quantitative objectives. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 140–156. Springer, Heidelberg (2009)

[BGHJ09] Bloem, R., Greimel, K., Henzinger, T., Jobstmann, B.: Synthesizing robust systems. In: Proc. Formal Methods in Computer Aided Design (FMCAD), pp. 85–92 (2009)

[BGJ+07] Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Automatic hardware synthesis from specifications: A case study. In: Proceedings of the Design, Automation and Test in Europe, pp. 1188–1193 (2007)

[CHP07] Chatterjee, K., Henzinger, T.A., Piterman, N.: Generalized parity games. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 153–167. Springer, Heidelberg (2007)

[dAF07] de Alfaro, L., Faella, M.: Accelerated algorithms for 3-color parity games with an application to timed games. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 108–120. Springer, Heidelberg (2007)

[Dij68] Dijkstra, E.W.: Cooperating sequential processes. In: Genuys (ed.) Programming Languages, pp. 43–112. Academic Press, London (1968)

[Dij74] Dijkstra, E.: Self-stabilizing systems in spite of distributed control. Communications of the ACM 17, 643–644 (1974)

[EJ91] Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Proc. 32nd IEEE Symposium on Foundations of Computer Science, October 1991, pp. 368–377 (1991)

[EKA08] Ebnenasir, A., Kulkarni, S.S., Arora, A.: Ftsyn: a framework for automatic synthesis of fault-tolerance. Software Tools for Technology Transfer 10, 455–471 (2008)

[EL86] Emerson, E.A., Lei, C.-L.: Efficient model checking in fragments of the propositional mu-calculus. In: Proceedings of the First Annual Symposium of Logic in Computer Science, June 1986, pp. 267–278 (1986)

[FD08] Fey, G., Drechsler, R.: A basis for formal robustness checking. In: ISQED, pp. 784–789 (2008)

[GBJV08] Greimel, K., Bloem, R., Jobstmann, B., Vardi, M.: Open implication. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 361–372. Springer, Heidelberg (2008)

[JGWB07] Jobstmann, B., Galler, S., Weiglhofer, M., Bloem, R.: Anzu: A tool for property synthesis. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 258–262. Springer, Heidelberg (2007)

[Jur00] Jurdziński, M.: Small progress measures for solving parity games. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 290–301. Springer, Heidelberg (2000)

[KE05] Kulkarni, S.S., Ebnnenasir, A.: Complexity issues in automated synthesis of failsafe fault-tolerance. IEEE Transactions on Dependable and Secure Computing 2, 1–15 (2005)

[PP06] Piterman, N., Pnueli, A.: Faster solutions of Rabin and Streett games. In: Logic in Computer Science, pp. 275–284 (2006)

[PPS06] Piterman, N., Pnueli, A., Saár, Y.: Synthesis of reactive(1) designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2005)

[Zie98] Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science 200(1-2), 135–183 (1998)