Synthesizing Robust Systems

Roderick Bloem*, Karin Greimel*, Thomas A. Henzinger†‡, and Barbara Jobstmann†
*Graz University of Technology, †EPFL, ‡IST Austria

***Abstract*—Many specifications include assumptions on the environment. If the environment satisfies the assumptions then a correct system reacts as intended. However, when the environment deviates from its expected behavior, a correct system can behave arbitrarily. We want to synthesize *robust* systems that degrade gracefully, i.e., a small number of environment failures should induce a small number of system failures. We define *ratio games* and show that an optimal robust system corresponds to the winning strategy of a ratio game, where the system minimizes the ratio of system errors to environment errors. We show that ratio games can be solved in pseudopolynomial time.**

I. Introduction

Suppose that a system is required to accept up to 1000 requests per second and to respond to each request within 0.1 seconds. What should the system do when request number 1001 arrives? There are several options, including terminating the system, dropping the extra request, or delaying a response. Clearly, all of these approaches satisfy the specification, but some are better than others. (Cf. [1].)

The formal functional specifications used in Design Automation typically only describe the behavior of a system in absence of environment failures. That is, (parts of) the specification are of the form $A \rightarrow G$, where A is an environment assumption and G is a guarantee. This approach leaves the behavior of the system unspecified when A is not fulfilled and neither verification tools nor synthesis tools take such behavior into account. In practice, however, the environment may fail, due to incomplete specifications, operator errors, faulty implementations, transmission errors, and the like. Thus, a system should not only be correct, it should also be *robust*, meaning that it "behaves 'reasonably,' even in circumstances that were not anticipated in the requirements specification [...]" [2].

For instance, consider the following informal specification of an arbiter. Initially, both input r (for request) and output g (for grant) are low. If the environment raises r, the system will eventually raise g. The environment is not allowed to lower r before g is raised. After g is raised, r must be lowered eventually, after which g is lowered. The obvious formalization of this specification does not have any requirements on the behavior of the system if the environment lowers a request too early. In fact, if this ever occurs, the system can act arbitrarily from that time on. This is clearly unreasonable, because the system can fulfill all its requirements even in this case. In general, of course, the system may have to fail in some way if the environment does. However, we prefer graceful degradation: the system error should increase slowly with the environment error.

This research was supported by the Swiss National Science Foundation (Indo-Swiss Research Program and NCCR MICS) and the European Union projects ArtistDesign, COMBEST, and COCONUT.

This paper proposes a formal notion of robustness through graceful degradation for discrete functional safety properties: A small error by the environment should induce only a small error by the system, where the error is defined quantitatively as part of the specification, for instance, as the number of failures. Given such a specification, we define a system to be robust if a finite environment error induces only a finite system error. As a more fine-grained measure of robustness, we define the notion of k-robustness, meaning that on average, the number of system failures is at most k times larger than the number of environment failures. We show that the synthesis question for robust systems can be solved in polynomial time as a one-pair Streett game and that the synthesis question for k-robust systems can be solved using *ratio games*. Ratio games are a novel type of graph games in which edges are labeled with a cost for each player, and the aim is to minimize the ratio of the sum of these costs. We show that ratio games are positional, that the associated decision problem is in NP $\cap$ co-NP, and that they can be solved in pseudopolynomial time. They can be solved in polynomial time if the cost of a failure is assumed to be constant.

Section II fixes the notation used in the paper. In Section III, we present our framework based on error functions, and define robustness and k-robustness. In Section IV, we introduce ratio games and show how to solve them. Section V shows how to use ratio games to construct correct and robust systems. We present related work in Section VI and conclude in Section VII.

II. Preliminaries

For a word $w = w_1 \ldots$, let $|w| \in \mathbb{N} \cup \{\infty\}$ be the length of the word and let $w[..i] = w_1 \ldots w_i$ be the prefix of length i. We denote the set of all finite (infinite) words over the alphabet A by A^* (A^ω).

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 $\Sigma = 2^{AP}$.

Moore machines: We use Moore machines to represent systems. A *Moore machine* with input alphabet Σ_I and output alphabet Σ_O is a tuple $M = (Q, q_0, \delta, \lambda)$, where Q is the set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma_I \rightarrow Q$ is the transition function, and $\lambda : Q \rightarrow \Sigma_O$ is the output function. In each state, the Moore machine outputs a letter in

Σ_O, then reads a letters in Σ_I, and moves to the next state. The *run* of M on a sequence $x = x_0 x_1 \ldots \in {\Sigma_I}^\omega$ is a sequence $\rho_0\rho_1 \ldots \in Q^\omega$, where $\rho_0 = q_0$ and $\rho_{i+1} = \delta(\rho_i, x_i)$. The corresponding *word* is $\lambda(\rho) = w_0 w_1 \ldots \in \Sigma^\omega$, where $w_i = \lambda(\rho_i) \cup x_i$. The *language of* M, $L(M) \subseteq \Sigma^\omega$, consists of the words corresponding to the runs of M. We define $L^*(M) = L(M) \cap \Sigma^*$.

Automata: A complete deterministic *automaton* over the alphabet Σ is a tuple $A = (Q, q_0, \delta)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\delta : Q \times \Sigma \to Q$ is the transition function. A *run* of an automaton A on a word $w = w_0 w_1 \ldots \in \Sigma^* \cup \Sigma^\omega$ is the longest sequence $\rho(w) = \rho_0\rho_1 \ldots \in Q^* \cup Q^\omega$ such that $\rho_0 = q_0$, and $\rho_{i+1} = \delta(\rho_i, w_i)$. The *product automaton* $A = A_1 \times A_2$ of two automata is defined as usual.

A *safety automaton* $A = (Q, q_0, \delta, F)$ is a complete deterministic automaton (Q, q_0, δ) together with a set $F \subseteq Q$ of *accepting states* such that there are no edges from non-accepting to accepting states.An infinite run is *accepting* if it never leaves F. An automaton accepts a word if its run is accepting. We call the set $L(A)$ of infinite words accepted by A the *language of* A.

Specifications: We use safety automata to specify the desired behavior of a Moore machine. Given a safety automaton A, we say the Moore machine M *satisfies* A, if $L(M) \subseteq L(A)$. In our examples, we also show LTL formulas describing the discussed properties. Readers familiar with LTL [3] will find them useful, while they can be safely ignore by readers not familiar with LTL.

Single and Double Cost Automata: A *single (double) cost automaton* over the alphabet Σ is a tuple $C = (Q, q_0, \delta, c)$ consisting of a complete deterministic automaton (Q, q_0, δ) and a cost function $c : Q \times \Sigma \to \mathbb{N}$ ($c : Q \times \Sigma \to \mathbb{N} \times \mathbb{N}$, respectively) that associates to each transition a value in $\mathbb{N}$ ($\mathbb{N} \times \mathbb{N}$, resp.) called *cost*. In a double cost automaton, we use c_s and c_e to refer to the cost function of the first and the second component, respectively. The *maximal cost* is the smallest $W \in \mathbb{N}\ \forall q \in Q, \sigma \in \Sigma : c(q, \sigma) \leq W$ ($c_e(q, \sigma), c_s(q, \sigma) \leq W$). The cost of a word $w \in \Sigma^* \cup A^\omega$, denoted by $C(w)$, is the sum $\sum_{i=0}^{|W|} c(\rho(w)_i, w_i)$, For double cost automata, we use $C_e(w)$ and $C_s(w)$ to refer to the first and second component, respectively, of the cost of the word w.

The *sum of two cost automata* $A_1 = (Q_1, q_{01}, \delta_1, c_1)$ and $A_2 = (Q_2, q_{02}, \delta_2, c_2)$ is the cost automaton $A = A_1 + A_2 = (Q, q_0, \delta, c)$, where A is the product of the automata A_1 and A_2 with costs $c = c_1 + c_2$, i.e., $c((q_1, q_2), \sigma) = c_1(q_1, \sigma) + c_2(q_2, \sigma)$. The *product of two single cost automata* $A_1 = (Q_1, q_{01}, \delta_1, c_1)$ and $A_2 = (Q_2, q_{02}, \delta_2, c_2)$ is a double cost automaton $A = A_1 \times A_2 = (Q, q_0, \delta, c)$, where A is the product of the automata A_1 and A_2 with costs $c = (c_1, c_2)$, i.e., $c((q_1, q_2), \sigma) = (c_1(q_1, \sigma), c_2(q_2, \sigma))$.

Games: 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 $(s_i, s_{i+1}) \in E$. We denote the set of all plays by Ω. Given a game graph $G = (S, E)$, a *strategy* for Player 1 is a function $\pi_1 : S^* S_1 \to S$ mapping a sequence of states ending in a Player-1 state to a successor state such that for all $s_0 \ldots s_i \in S^* S_1$, we have that $(s_i, \pi_1(s_0 \ldots s_i)) \in E$. A Player-2 strategy is defined similarly. We denote by Π_1 and Π_2 the set of all possible Player-1 and Player-2 strategies, respectively. A strategy is *positional* if it depends only on the current state. We present a positional strategy π_p as a function from S_p to S. Let $\rho(\pi_1, \pi_2, s)$ denote the unique play starting at s when Player 1 plays the strategy π_1 and Player 2 plays π_2.

The value of a play is given by a *value function* $v : \Omega \to \mathbb{R} \cup \{-\infty, \infty\}$. The value of a state s under strategy π_1 and π_2, denoted by $v(\pi_1, \pi_2, s)$, is the value of the play $\rho(\pi_1, \pi_2, s)$We consider complementary objectives for the two players: Player 1 tries to minimize the value of a state and Player 2 tries to maximize it. (Note that the converse is more usual.) The Player-1 value of a state s under the strategy π_1 is $\sup_{\pi_2 \in \Pi_2}(v(\pi_1, \pi_2, s))$. A strategy π_1 is optimal for Player 1 in state s if the Player-1 value of the state s under the strategy π_1 is minimal. The Player-2 value and Player-2 optimal strategies are defined correspondingly. The value of a state s denoted by $v(s)$ is the Player-1 value of the play starting in s, in which both players play optimally.

A *game* is a game graph together with a value function. The game graph defines the possible actions of the players. The value function describes the objectives of the players. A *mean payoff game* is describe as a tuple $((S, s_0, E), w)$, where (S, s_0, E) is a game graph and $w : E \to \mathbb{N}$ is a *payoff function*. The value function for a play $\rho = s_0 s_1 \ldots$ in a mean payoff game is $v(\rho) = \limsup_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n} w(e_i)$ with $e_i = (s_i, s_{i+1})$. A *one-pair Streett game* is a tuple $((S, s_0, E), F_1, F_2)$ consisting of a game graph (S, s_0, E) and two sets $F_1, F_2 \subseteq S$, where $v(\rho) = 0$ iff $(\forall i \geq 0\ \exists j \geq i : s_j \in F_1) \to (\forall i \geq 0\ \exists j \geq i : s_j \in F_2)$. We say a Streett game is *winning for Player 1* if the value of the initial state s_0 is 0.

An automaton $A = (Q, q_0, \delta)$ over the alphabet Σ can be translated into a game graph (S, s_0, E) as follows. We define the set of Player-1 states as $S_1 = \{s_{(q,\sigma_i)} \mid q \in Q \text{ and } \sigma_i \in \Sigma_I\} \cup \{s_0\}$. The Player-2 states S_2 are given by the set $S_2 = \{s_{(q,\sigma_o)} \mid q \in Q \text{ and } \sigma_o \in \Sigma_O\}$. The set of game states is the set $S = S_1 \cup S_2$. Every state of the game (except for the initial state) represents a state of the automaton and an input or output label. Note that this corresponds to moving from a transition-labeled to a state-labeled system. Every outgoing transition of a state q in A is translated into two steps of the game: first, Player 1 chooses a letter σ_o from Σ_O by moving to the states $s_{(q,\sigma_o)}$, then Player 2 chooses a letter σ_i from Σ_I and moves according to the transition relation to a new state $s_{(q',\sigma_i)}$ such that $\delta(q, \sigma_o \cup \sigma_i) = q'$. Formally, we have that $E_1 = \{(s_{(q,\sigma_i)}, s_{(q,\sigma_o)}) \mid q \in Q, \sigma_o \in \Sigma_O, \text{ and } \sigma_I \in$

$\Sigma_I\} \cup \{(s_0, s_{q_o,\sigma_o}) \mid \sigma_o \in \Sigma_O\}$, $E_2 = \{(s_{(q,\sigma_o)}, s_{(q',\sigma_I)}) \mid q, q' \in Q, \sigma_o \in \Sigma_O, \sigma_I \in \Sigma_I,$ and $\delta(q, \sigma_o \cup \sigma_i) = q'\}$, and $E = E_1 \cup E_2$.

III. Defining Robustness

In this section we introduce our notion of robustness based on error specifications. We show how error specifications relate to classical specifications and the notion of realizability. We conclude with an example.

Definition 1. *An* error function *is a function $d : \Sigma^* \cup \Sigma^\omega \to \mathbb{N} \cup \{\infty\}$. The function is monotonically increasing in the sense that if w' is a prefix of w then $d(w') \leq d(w)$.*

An error specification *is a pair of error functions (d_e, d_s).*

The error functions define a distance between allowed and observed behavior, for instance, by measuring the number of failures in some appropriate sense. Thus, $d(w) = 0$ indicates that w fulfills the specification, and a higher value indicates a more serious violation of the specification. Error specifications provide a measure of "badness" for both the environment behavior (using d_e) and the system behavior (using d_s) and form the specifications we use in the sequel. We assume that these specifications are provided by the user.

Definition 2. *A Moore machine M* realizes *an error specification (d_e, d_s) if $\forall w \in L(M) : d_e(w) = 0$ implies $d_s(w) = 0$.*

Thus, an error specification induces a classical specification $A \to G$, where $A = \{w \in \Sigma^\omega \mid d_e(w) = 0\}$ and $G = \{w \in \Sigma^\omega \mid d_s(w) = 0\}$ are sets of infinite words.

The following notion is an alternative to realizability, forbidding the system to make mistakes before the environment does.

Definition 3. *A Moore machine M* strictly realizes *an error specification (d_e, d_s) if $\forall w \in L^*(M) : d_e(w[..|w|-1]) = 0$ implies $d_s(w) = 0$. An error specification is* strictly realizable *if there exists a Moore machine that strictly realizes it.*

Example 4. *An example of a specification that is realizable but not strictly realizable is $A_1 \wedge A_2 \to G_1 \wedge G_2$, where x is an input, y is an output, A_1 requires that x is always true ($\mathsf{G}\,x$), A_2 says that x is initially equal to y ($x \leftrightarrow y$), G_1 states that y is always true ($\mathsf{G}\,y$), and G_2 states that x in the first step and y in the second step are different ($x \not\leftrightarrow (\mathsf{X}\,y)$). All Moore machines that realize the specification start with setting y to false, which violates the guarantees but forces the environment to do the same*[1].

Definition 5. *A Moore machine M is* robust *with respect to an error specification (d_e, d_s) if $\forall w \in L(M) : d_e(w) \neq \infty$ implies $d_s(w) \neq \infty$.*

This means that a robust system can recover from a finite environment error. Note that a system can be robust with respect to a specification that it does not realize if it contains a word with a finite system error but no environment error. Error specifications can forbid words by assigning infinite system costs. (In particular, this is possible when such specifications are given by double cost automata, as below.)

[1]This specification is based on an example by Marco Roveri.

In order to calculate the quality of a robust system we want to calculate the largest system error for every environment error.

Definition 6. *A Moore machine M is k-robust with respect to an error specification (d_e, d_s) if $\exists d \in \mathbb{N} : \forall w \in L^*(M) : d_s(w) \leq k \cdot d_e(w) + d$.*

Obviously, every k-robust system is robust, regardless of k. Also, every robust system is k-robust for some finite k, see Theorem 15, i.e., for every finite Moore machine, the growth of the system error is either linear with respect to the environment error or unbounded. This motivates our choice of the robustness measure as a linear function. The definition of k-robustness allows us to rank Moore machines with respect to error specifications: A smaller k is better, it means that the system error increases slowly with the environment error. The constant d allows the system finitely many system failures independent of the environment error. In this paper, we focus on the infinite behavior of a machine, and note that d can be bounded by the product of the size of the Moore machine and the maximal weight. We leave minimization of d to future work.

Definition 7. *A Moore machine* (k-)robustly (and strictly) realizes *an error specification if it (strictly) realizes the specification and it is (k-)robust with respect to the specification.*

In the remainder, we use double cost automata to define error specifications. The environment (system) error function associated with C maps each $w \in \Sigma^* \cup \Sigma^\omega$ to its cost $C_e(w)$ ($C_s(w)$, respectively). Note that a double cost automaton can be seen as the product of two single cost automata. We can construct an error specification from a set of cost automata C_{A_i} for the system and C_{G_i} for the environment. The error specification (a double cost automaton) is the product of the sum of all C_{A_i} and the sum of all C_{G_i}.

Example 8. *Consider a system with two request signals r_1 and r_2 as inputs and two grant signals g_1 and g_2 as outputs. We want the system to respond to each request with a grant in the next step. Formally, we require that the system satisfies $G_i = \mathsf{G}(r_i \to \mathsf{X}\,g_i)$ for $i \in \{1, 2\}$. The system should also guarantee that grants are mutually exclusive, i.e., $G_3 = \mathsf{G}\,\neg(g_1 \wedge g_2)$. To avoid a contradicting specification, we assume that requests are also mutually exclusive, i.e., $A = \mathsf{G}\,\neg(r_1 \wedge r_2)$. Figure 1 shows two safety automata, one for A and one for G_1 and G_2. Note that we summarize labels on edges with Boolean expressions over r_i and g_i, where a horizontal alignment of two variables represents a conjunction and two vertically aligned variables are disjoint. We use a bar to denote negation and $\top$ to denote true. States depicted with two cycles are accepting states. Note that the automaton for G_3 is exactly the same as for A, where r_1 and r_2 are renamed to g_1 and g_2, respectively.*

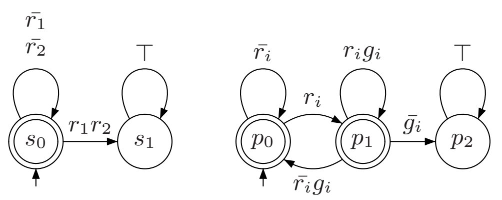
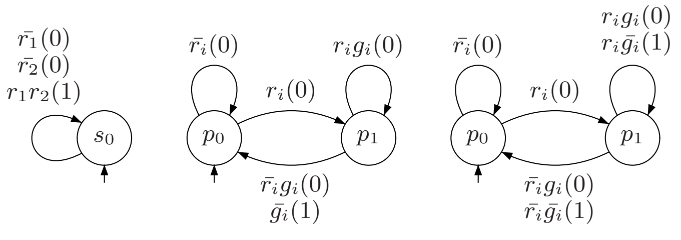


Fig. 1. Automata for $A = \mathsf{G}(\neg(r_1 \wedge r_2))$ and $G_i = \mathsf{G}(r_i \rightarrow \mathsf{X}\, g_i)$.



(a) C_A (b) C_{G_i} (c) C'_{G_i}

Fig. 2. Cost automata counting violations of A and G_i, respectively.

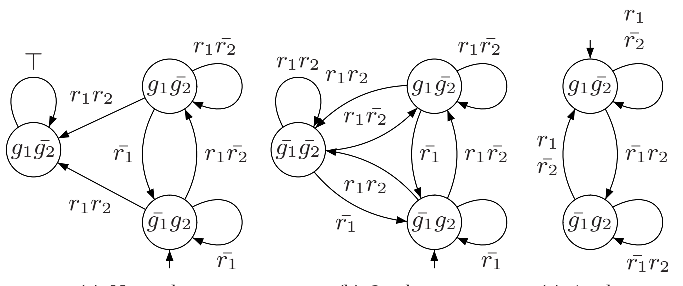
Starting from the specification $A \rightarrow (G_1 \wedge G_2 \wedge G_3)$, we can define what it means for the system and the environment to fail. In particular, the environment violates assumption A if it raises r_1 and r_2 at the same time. This corresponds to taking the edge from s_0 to s_1. In Figure 2(a), we show a cost automaton that counts every violation of the environment. Note that once the environment "pays" for taking the edge $r_1 r_2$, we go back to the initial state, resetting the specification. Similarly, if the system violates Guarantee G_i by choosing to go from p_1 to p_2, it also incurs cost 1 as shown in Figure 2(b).

Note that it is up to the user to define the cost of a violation and the state in which to continue after the specification is violated. A reset or a skip are two natural alternatives. A reset corresponds to an edge to the initial state. For a skip, we simply add a self-loop. In Figure 2(c) we show an alternative cost automaton for G_i with $i \in \{1, 2\}$, which uses a mixture of reset and skip. For the cost automaton C_{G_i}, the word $(r_1, \bar{g_1})(r_1, \bar{g_1})(\bar{r_1}, \bar{g_1})^\omega$ has cost 1 whereas it has cost 2 for the cost automaton C'_{G_i}. For the second automaton, the cost corresponds to the number of unanswered requests.

The costs on the edges are given by the user. For instance, the user might consider a violation of the mutual-exclusion properties G_3 more severe and associate with it a higher cost than a violation of the response properties G_1 or G_2.

Given cost automata C_{G_1}, C_{G_2}, and C_{G_3} that describe the cost and the behavior associated with a violation of the corresponding property (cf. Fig. 2), we can construct a cost automaton $C_G = C_{G_1} + C_{G_1} + C_{G_1}$ for $G = G_1 \wedge G_2 \wedge G_3$. The automaton C_G defines the error function of the system. The cost automaton for the environment C_A (cf. Fig. 2(a)) specifies the error function of the environment. The product $C = C_A \times C_G$ is the error specification.

Figure 3(a) shows a system M (synthesized with Lily *[4]) for the specification $A \rightarrow G$. It is easy to see that M satisfies $A \rightarrow G$. As long as the environment satisfies A, which means that it does not provide r_1 and r_2 simultaneously, the system responds to each r_i with the corresponding g_i in the next step. However, M is not robust with respect to C: The input sequence $i = (r_1 r_2)(\bar{r_1} r_2)^\omega$ has cost one, but the output of the system has cost ∞.*



(a) Non-robust (b) 2-robust (c) 1-robust

Fig. 3. A non-robust, 2-robust, and a 1-robust system.

Figure 3(b) and 3(c) show two systems that are robust with respect to the error specification, for any word with finitely many environment errors the systems produce finitely many system errors. The system in Figure 3(b) is 2-robust with respect to the error specification whereas the system in Figure 3(c) is 1-robust. For the input $(r_1 r_2)^\omega$ the output of the first Moore machine is $(\bar{g_1} g_2)(\bar{g_1}\bar{g_2})^\omega$ and for the second it is $(g_1 \bar{g_2})^\omega$.

Note that out of the three systems in Figure 3 (which all satisfy $A \rightarrow G$) the system in Figure 3(c) is the most robust one. In our opinion, it is also the one most likely to please the designer.

In Section V we show how to synthesize (strictly) realizing robust and k-robust systems from an error specification. We also show how these notions can be verified. The next section introduces Ratio games, which are crucial to our synthesis algorithms.

IV. Ratio Games

In this section we introduce ratio games, which we need to synthesize k-robust systems. Intuitively, a system is k-robust if the ratio of the system error to the environment error is smaller than or equal to k for every word of the system. An optimal strategy for Player 1 in a ratio game minimizes this ratio.

Definition 9. *A* ratio game[2] *G is a tuple $((S, s_0, E), w_1, w_2)$ consisting of a game graph (S, s_0, E) and two weight functions $w_1, w_2 : E \rightarrow \mathbb{N}$ mapping edges to non-negative integer values. The value function for a play $\rho = s_0 s_1 \ldots \in S^\omega$ is*

$$v(\rho) = \lim_{m\rightarrow\infty} \limsup_{l\rightarrow\infty} \frac{\sum_{i=m}^{l} w_1(s_i, s_{i+1})}{1 + \sum_{i=m}^{l} w_2(s_i, s_{i+1})} \qquad (1)$$

Ratio games are a generalization of mean payoff games. If $w_2(e) = 1$ for all $e \in E$, then G is a mean payoff game. Note that the sequence of quotients for $l \rightarrow \infty$ might diverge, which requires the use of $\limsup$ or $\liminf$. We follow the definition of mean payoff games and take the $\limsup$. The outer-most limit ensures that only the infinite behavior is relevant as in the definition of k-robustness, i.e., if $\sum_{i=0}^{\infty} w_1(e_i)$ is finite, then $v(\rho) = 0$. The addition of 1 in the denominator avoids

[2]Our graph-based ratio games should not be confused with those of [5], which represent games in a normal form, enumerating all strategies. We cannot use that representation to obtain a polynomial algorithm.

division by zero. It does not influence the value of $v(\rho)$ if $\sum_{i=0}^{\infty} w_2(e_i)$ is infinite.

The *maximal weight* W in a ratio game $((S, s_0, E), w_1, w_2)$ is defined by $W = \max\{w_i(e) \mid e \in E,\ i \in \{1, 2\}\}$. Note that the value $v(\rho)$ of a play ρ, where both players play positional strategies, is in the set $V = \{0, \frac{1}{|S|\cdot W}, \dots, \frac{|S|\cdot W}{1}, \infty\}$. Lemma 10 shows that ratio games have optimal positional strategies, which implies that it suffices to consider positional strategies and that the value of every state is in V.

Lemma 10. *Ratio games have optimal positional strategies.*

Proof: It suffices to show that the two one-player games ($S_2 = \emptyset$, respectively $S_1 = \emptyset$) have optimal positional strategies [6]. Consider a game graph G with $S_2 = \emptyset$. Take in G a simple cycle with the minimum ratio r of all simple cycles.We show that the positional strategy π_1 that goes to this simple cycle and stays within it forever is optimal. Note that the value $v(\rho)$ of the play ρ induced by the strategy π_1 is r, since the outer-most limit in Eq. 1 allows us to ignore a finite prefix of ρ. If $r = 0$, the claim trivially holds. If $r = \infty$, then in any simple cycle the sum of the weights w_2 is 0 and the sum of the weights w_1 is strictly greater than 0. This implies that all edges e on cycles have weight $w_2(e) = 0$ and in every cycle there is at least one edge e with $w_1(e) > 0$, and so any infinite play has ratio ∞. For $0 < r < \infty$, let r be $\frac{a}{b}$ for some integers $a, b > 0$ and let ρ' be an arbitrary play in the single player game. We decompose ρ' into a sequence of ratios $\frac{a_1}{b_1}, \frac{a_2}{b_2} \dots$ by the following procedure (cf. [7]): we put the states of ρ' on a stack in the order of their appearance, once we encounter a state q that is already on the stack, we remove the sequence from the first to the second appearance of q and compute its ratio $\frac{a_i}{b_i}$. Note that the sum of the weights w_1 and w_2 in this cycle can be c_i-times larger than a_i and b_i, respectively, where c_i is some integer constant between 1 and $W \cdot |S|$. Note that the height of the stack is at most $|S|$. Due to the outer-most limit, we can ignore the part of ρ' that is always left on the stack in the computation of the value $v(\rho')$. Then, $v(\rho') = \limsup_{l\to\infty} \frac{\sum_{i=1}^{l} c_i\cdot a_i}{1+\sum_{i=1}^{l} c_i\cdot b_i}$ for some constants $0 < c_i \leq W \cdot |S|$. Since the minimum simple-cycle ratio is $\frac{a}{b}$, we know that $\frac{a_i}{b_i} \geq \frac{a}{b}$ for all $i > 0$ and together with the fact that c_i's are positive integer constants, we know that $v(\rho') \geq \limsup_{l\to\infty} \frac{\sum_{i=1}^{l} a}{1+\sum_{i=1}^{l} b}$ and therefore $v(\rho') \geq \frac{a}{b}$. The proof for Player-2 games is analogous. ■

The decision problem of a ratio (mean payoff) game is, given a ratio r (mean payoff v) decide if the value of the game is at least r (v). The decision problem for mean payoff games is in NP ∩ co-NP [7]. We show how the decision problem for ratio games can be reduced to the decision problem of mean payoff games. The reduction shows that the decision problem for ratio games is in NP ∩ co-NP. We also use this reduction to calculate the values of the states in a ratio game. The reduction is similar to that used by Lawler [8] for the reduction of ratio graphs to the minimal mean cycle problem.

Lemma 11. *Let $G_R = ((S, s_0, E), w_1, w_2)$ be a ratio game with maximal weight W. Given a ratio $\frac{a}{b}$ with $0 \leq a \leq |S|\cdot W$ and $0 < b \leq |S|\cdot W$, we can decide whether a state has value $v = \frac{a}{b}$, $v < \frac{a}{b}$, or $v > \frac{a}{b}$ in $O(|S|^3 \cdot W^2 \cdot |E|)$ time.*

Proof: We reduce the decision for the ratio game to a decision for the mean payoff game $G_{\text{MP}} = ((S, s_0, E), w)$ with payoff function $w(e) = b \cdot w_1(e) - a \cdot w_2(e)$. In the following, let v_{R} ($v_{\text{R}}(\rho)$) be the value (of run ρ) in G_{R} and similarly for v_{MP}.

We show that $v_{\text{R}} \leq \frac{a}{b}$ implies $v_{\text{MP}} \leq 0$ and $v_{\text{R}} \geq \frac{a}{b}$ implies $v_{\text{MP}} \geq 0$. The decision whether $v_{\text{MP}} < 0$, $v_{\text{MP}} = 0$, or $v_{\text{MP}} > 0$ can be made in $O(|S|^2 \cdot W' \cdot |E|)$ time, where W' is the maximal weight in the mean-payoff game [7]. We have $W' \leq b \cdot W \leq |S| \cdot W^2$, thus the decision for the ratio game can be made in $O(|S|^3 \cdot W^2 \cdot |E|)$ time.

Suppose $v_{\text{R}} \leq \frac{a}{b}$. We show that Player 1 can achieve a run of value at most 0 in G_{MP} and thus $v_{\text{MP}} \leq 0$. Let π_1 be a positional optimal Player-1 strategy for G_{R} and let π_2 be a positional optimal strategy for Player 2 in G_{MP}. Because both strategies are positional, $\rho(s_0, \pi_1, \pi_2)$ consists of a stem and a simple cycle, say $\rho = (e'_1, \dots, e'_m) \cdot (e_1, \dots, e_n)^\omega$. Note that $v_{\text{R}}(\rho) = \frac{\sum_{i=0}^{n} w_1(e_i)}{\sum_{i=0}^{n} w_2(e_i)}$ and $v_{\text{MP}}(\rho) = \frac{b\sum_{i=0}^{n} w_1(e_i) - a\sum_{i=0}^{n} w_2(e_i)}{n}$. Suppose $\sum_{i=0}^{n} w_1(e_i) > 0$, then, since $v_{\text{R}} \leq \frac{a}{b}$ and is thus finite, we have $\sum_{i=0}^{n} w_2(e_i) > 0$. It follows that $\frac{\sum_{i=0}^{n} w_1(e_i)}{\sum_{i=0}^{n} w_2(e_i)} \leq \frac{a}{b}$ implies $\frac{b\sum_{i=0}^{n} w_1(e_i) - a\sum_{i=0}^{n} w_2(e_i)}{n} \leq 0$. If $\sum_{i=0}^{n} w_1(e_i) = 0$, then $\frac{b\sum_{i=0}^{n} w_1(e_i) - a\sum_{i=0}^{n} w_2(e_i)}{n} = \frac{-(a\sum_{i=0}^{n} w_2(e_i))}{n} \leq 0$.

The proof that $v_{\text{R}} \geq \frac{a}{b}$ implies that $v_{\text{MP}} \geq 0$ is similar, using an optimal strategy for Player 2 in G_{R}. ■

Theorem 12. *Given a ratio game $((S, E), w_1, w_2)$ with maximal weight W, the value for every $s \in S$ can be computed in $O(|S|^3 \cdot W^2 \cdot |E| \cdot \log(|S| \cdot W))$.*

Proof: We use the decision procedure from Lemma 11 to perform a binary search on the list of possible values $V \setminus \{\infty\}$. If the ratio is greater than $|S| \cdot W$, it is infinite. There are less than $(|S| \cdot W)^2$ different ratios, thus we need at most $2 \cdot \log(|S| \cdot W)$ calls to the decision procedure. ■

Given an algorithm to find the values of the game we can use the "group testing" technique from [7] to find optimal positional strategies.

Theorem 13. *Given a ratio game $((S, E), w_1, w_2)$ with maximal weight W, positional optimal strategies for both players can be found in $O(|S|^4 \cdot \log(\frac{|E|}{|S|}) \cdot |E| \cdot \log(|S| \cdot W) \cdot W^2)$.*

All our ratio game algorithms are polynomial in the size of the game graph but pseudopolynomial in the weights. They are polynomial if $W = 1$.

V. Verifying and Synthesizing Robust Systems

This section describes the verification and synthesis algorithms for robust systems. First, we establish the correlation between the ratio in Definition 9 and k-robustness.

Any error specification C with cost functions c_e and c_s can be translated into a ratio game G. The weight functions w_1 and w_2 are given by the cost functions c_s and c_e respectively. Formally, $w_{1(2)}((s_{(q,\sigma_i)}, s_{(q,\sigma_o)})) = 0$ and $w_{1(2)}((s_{(q,\sigma_o)}, s_{(q',\sigma_i)})) = c_{s(e)}(q, \sigma_o \cup \sigma_i)$, where $(s_{(q,\sigma_i)}, s_{(q,\sigma_o)}) \in E_1$ and $(s_{(q,\sigma_o)}, s_{(q',\sigma_i)}) \in E_2$ (see Section II). Every play $\rho_G = s_0, s_{(q_0,\sigma_o)}, s_{(q',\sigma_i)}, s_{(q',\sigma'_o)} \dots$ of G corresponds to a run $\rho_C = q_0 q' \dots$ of C on $w = (\sigma_o, \sigma_i)(\sigma'_o, \sigma'_i)$.

Lemma 14. *Given a Moore machine M and an error specification C with cost function c_e and c_s, M is k-robust iff for all words $w \in L(M)$, the run $\rho(w) = q_0 \dots$ of C on $w = w_0 \dots$ satisfies*

$$v(w) = \lim_{m\to\infty} \limsup_{l\to\infty} \frac{\sum_{i=m}^{l} c_s(q_i, w_i)}{1 + \sum_{i=m}^{l} c_e(q_i, w_i)} \le k. \qquad (2)$$

Proof: If there exists a $d \in \mathbb{N}$ such that for all finite prefixes $w' = w_0 \dots w_n$ of w we have $\sum_{i=0}^{n} c_s(q_i, w_i) \le k \cdot \sum_{i=0}^{n} c_e(q_i, w_i) + d$, then $\frac{\sum_{i=0}^{n} c_s(q_i,w_i)}{1+\sum_{i=0}^{n} c_e(q_i,w_i)} \le k + \frac{d-k}{1+\sum_{i=0}^{n} c_e(q_i,w_i)}$ holds as well. This implies that $\lim_{m\to\infty} \limsup_{l\to\infty} \frac{\sum_{i=m}^{l} c_s(q_i,w_i)}{1+\sum_{i=m}^{l} c_e(q_i,w_i)} \le k$ because $\limsup_{l\to\infty} \sum_{i=0}^{l} c_e(q_i, w_i)$ is either some finite value d' or infinite. In the first case, $\sum_{i=0}^{n} c_s(q_i, w_i) \le k \cdot d' + d$ for any $n \ge 0$. Therefore, $\limsup_{l\to\infty} \sum_{i=0}^{l} c_s(q_i, w_i)$ is also finite. Then, $\lim_{m\to\infty} \limsup_{l\to\infty} \frac{\sum_{i=m}^{l} c_s(q_i,w_i)}{1+\sum_{i=m}^{n} c_e(q_i,w_i)} = 0 \le k$. In the second case, $\lim_{m\to\infty} \limsup_{l\to\infty} \frac{d-k}{1+\sum_{i=m}^{l} c_e(q_i,w_i)}$ converges to 0.

For the other direction, consider the product CM of C and M. Then, for all $w \in L^*(M)$, $C_e(w) = CM_e(w) = \sum_{i=0}^{|w|-1} c_e(q_i, w_i)$ and $C_s(w) = CM_s(w) = \sum_{i=0}^{|w|-1} c_s(q_i, w_i)$, where $\rho_{CM}(w) = q_0 \dots q_{|w|}$ is the run of CM on w. Consider an arbitrary finite word $w \in L^*(M)$, if $|w| \le |C| \cdot |M|$, then $CM_s(w) \le |C| \cdot |M| \cdot W$ and $C_s(w) \le k \cdot C_e(w) + d$ holds for any $k \ge 0$ and $d = |C| \cdot |M| \cdot W$. Otherwise, if $|w| \ge |C| \cdot |M|$, we can decompose the run $\rho_{CM}(w)$ into simple cycles $c_1, \dots, c_m$ and a simple path p consisting of the remaining nodes. (See proof of Lemma 10.) Now consider the infinite words $u_1, \dots, u_m$ that correspond to the runs leading to the cycles $c_1, \dots, c_m$, respectively, and looping there forever. We know that $u_1, \dots, u_m$ are in $L(M)$ and, due to Eq. 2, that $v(u_j) \le k$ for all $1 \le j \le m$. Therefore, for every cycle, the sums of the weights c_e and c_s in the cycle, are either both 0 or their ratio is smaller or equal to k. Let $k = \frac{a}{b}$ and let $\frac{a_1}{b_1}, \dots, \frac{a_m}{b_m}$ be the ratios of the cycles whose ratio is nonzero, then $\sum_{i=0}^{|w-1|} c_s(q_i, w_i) = d' + \sum_{j=1}^{m} d_j \cdot a_j$ and $\sum_{i=0}^{|w-1|} c_e(q_i, w_i) = d'' + \sum_{j=1}^{m} d_j \cdot b_j$ for some $0 \le d', d'' \le |C| \cdot |M| \cdot W$ and $1 \le d_j \le |C| \cdot |M| \cdot W$. Using the fact that, if for all $1 \le j \le m$, $\frac{a_j}{b_j} \le \frac{a}{b}$ holds then $\sum_{j=1}^{m} d_j \cdot a_j \le \frac{a}{b} \sum_{j=1}^{m} d_j \cdot b_j$ holds, we obtain $\sum_{i=0}^{|w-1|} c_s(q_i, w_i) \le \frac{a}{b} \sum_{i=0}^{|w-1|} c_e(q_i, w_i) + d'$, which proves that M is k-robust. ■

A. Verification

We show that any robust system is k-robust.

Theorem 15. *If a Moore machine M with n_M states is robust with respect to an error specification C with n_C states and maximal system cost W, then M is $(n_C \cdot n_M \cdot W)$-robust.*

Proof: Let CM be the product of C and M. Lemma 14 shows that M is k robust if the ratio of all runs in CM is smaller or equal to k. Since one-player ratio games are positional (Lemma 10), the largest ratio corresponds to the largest ratio of a simple cycle in CM, which cannot be larger than $n_C \cdot n_M \cdot W$ because M is robust. ■

Next, we show how to verify if a given Moore machine is robust or k-robust.

Theorem 16. *Given a Moore machine M with n_M states, and an error specification C over the alphabet Σ, with n_C states and maximal cost W, we can decide if M is robust in $O(n_C \cdot n_M \cdot \Sigma)$ time. Given a k, we can check if M is k-robust in $O(n_C^3 \cdot n_M^3 \cdot \Sigma)$ time.*

Proof: Let CM be the product of C and M. M is not robust iff CM contains a cycle that contains an edge with nonzero system cost and no edge with nonzero environment cost. This can be checked in time linear in the number of edges in CM, which is $n_C \cdot n_M \cdot \Sigma$. We have that M is k-robust if the maximum simple cycle ratio in CM is smaller or equal to k. The maximum simple cycle ratio in a graph with n states and m transitions can be found in $O(n^2 \cdot m)$ time [9], thus we can find the maximum ratio in $O(n_C^3 \cdot n_M^3 \cdot \Sigma)$ time. ■

B. Synthesis

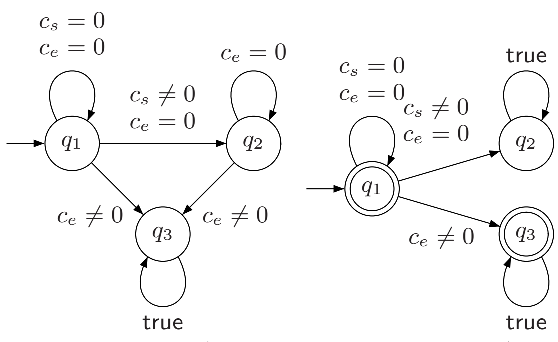
Next we show how to use Streett games to synthesize (strictly) realizing and robust systems and how to use ratio games to synthesize (strictly) realizing k-robust systems with optimal k.

Lemma 17. *Given an error specification C with n states and alphabet Σ, we can decide if a robust system exists in $O(n^2 \cdot \Sigma)$ time. If a robust system exists, it can be synthesized in $O(n^2 \cdot \Sigma)$ time.*

Proof: We translate the specification into a one-pair Streett game, F_1 is the set of states with incoming transitions with system costs and F_2 is the set of states with incoming transitions with environment costs. One-pair Streett games can be solved in $O(n \cdot m)$, where n is the number of states and m is the number of transitions [10]. ■

Theorem 18. *Given an error specification C with n states and alphabet Σ, we can decide if a robust and (strictly) realizing system exists in $O(n^2 \cdot \Sigma)$ time. The system can be synthesized in $O(n^2 \cdot \Sigma)$ time.*

Proof: In order to decide if a robust and realizing system exists, build the product automaton $CA_1 = (Q \times \{q_1, q_2, q_3\}, q_0, \delta, c)$ of the error specification C and the automaton A_1 shown in Figure 4(a). Let CA'_1 be CA_1, where the system costs of all transitions corresponding to the loop



(a) Automaton A_1. (b) Automaton A_2.

Fig. 4. Automata for calculating realizability and strict realizability

on state q_2 in Figure 4(a) are set to 1. Formally, the cost function of CA'_1 is $c'((q,x),\sigma) = (1, c_e((q,x),\sigma))$ if $x = q_2$ and $\delta((q,x),\sigma) = q_2$, and $c'((q,x),\sigma) = c((q,x),\sigma)$ in all other cases. Next, translate CA'_1 into a Streett game as above (proof of Lemma 17). A robust and realizing system exists iff the game is winning, and the winning strategy corresponds to a robust and realizing system.

First, assume there exists a winning strategy. No play in which Player 1 plays optimally visits a q_2-state infinitely often, because such a play has an infinite system error and zero environment error. Consequently, all words $w = (\sigma_o, \sigma_i)(\sigma'_o, \sigma'_i)\dots$ associated with a play $\rho = s_0, s_{(q_0,\sigma_o)}, s_{(q',\sigma_i)}, s_{(q',\sigma'_o)}\dots$ where Player 1 plays optimally satisfy $C_e(w) = 0$ implies $C_s(w) = 0$. Thus, the Moore machine corresponding to the winning strategy realizes the error specification. Second, assume there exists no winning strategy. A play where Player 2 plays optimally, has a finite environment cost and an infinite system cost. Either there exists no robust system or the play visits a q_2-state infinitely often. In the second case no system realizes the specification.

Similarly, to check for a robust and strictly realizing system, we build a Streett game from the product automaton CA'_2 of C and the automaton A_2 of Figure 4(b), where the system costs of all transitions corresponding to the loop on state q_2 are replaced by 1 and their environment costs are set to 0. Then, again any Player-1 optimal play avoids q_2-states. Consequently, for all words associated with a play where Player 1 plays optimally, all finite prefixes w' satisfy $C_e(w'[..|w'|-1]) = 0$ implies $C_s(w') = 0$. Thus, the Moore machine corresponding to a winning strategy strictly realizes the error specification. ■

Lemma 19. *Given an error specification C with n states, input alphabet Σ_I, output alphabet Σ_O, and maximal cost W, if a robust system exists, a k-robust system with minimal k can be synthesized in $O(n^5 \cdot (|\Sigma_I| + |\Sigma_O|)^4 \cdot \log(\frac{(|\Sigma_O|+n\cdot|\Sigma_I|)}{|\Sigma_I|+|\Sigma_O|}) \cdot (|\Sigma_O| + n \cdot |\Sigma_I|) \cdot \log(n \cdot (|\Sigma_I| + |\Sigma_O|) \cdot W) \cdot W^2)$.*

Proof: We synthesize k-robust systems with ratio games. The game graph is constructed from the double cost automaton C (see Section II). Lemma 14 shows that a positional strategy with value k corresponds to a k-robust Moore machine. An optimal positional strategy corresponds to a k-robust system with smallest possible k and $d \leq |C| \cdot W$.

The number of states in the game graph is $n\cdot|\Sigma_I|+n\cdot|\Sigma_O|$, the number of edges is $|E_1| + |E_2|$, where $|E_1| = n \cdot |\Sigma_O|$ and $|E_2| = n \cdot n \cdot |\Sigma_I|$. A winning strategy for Player 1 can be found in $O(n^4 \cdot (|\Sigma_I| + |\Sigma_O|)^4 \cdot \log(\frac{n\cdot(|\Sigma_O|+n\cdot|\Sigma_I|)}{n\cdot(|\Sigma_I|+|\Sigma_O|)}) \cdot n \cdot (|\Sigma_O| + n \cdot |\Sigma_I|) \cdot \log(n \cdot (|\Sigma_I| + |\Sigma_O|) \cdot W) \cdot W^2)$. ■

Theorem 20. *Given an error specification C with n states, input alphabet Σ_I, output alphabet Σ_O, and maximal cost W, if a robust and (strictly) realizing system exists, a k-robust system with minimal k that (strictly) realizes the specification can be synthesized in $O(n^5\cdot(|\Sigma_I|+|\Sigma_O|)^4\cdot\log(\frac{(|\Sigma_O|+n\cdot|\Sigma_I|)}{|\Sigma_I|+|\Sigma_O|})\cdot (|\Sigma_O| + n \cdot |\Sigma_I|) \cdot \log(n \cdot (|\Sigma_I| + |\Sigma_O|) \cdot W) \cdot W^2)$.*

Proof: For realizability translate CA'_1 from the proof of Theorem 18 into a ratio game. The system cost 1 for q_2-states guarantees that for any word w with $C_s(w) \neq 0$ and $C_e(w) = 0$ the ratio of the corresponding run has value ∞ in the ratio game. The ratios of other plays are not changed. If a play visits a q_2-state finitely often, the ratio is not influenced because we only look at the ratio in the limit.

For strict realizability translate CA'_2 from the proof of Theorem 18 into a ratio game. Since q_2-states have system cost 1 and environment cost 0, any run with a system failure before an environment failure has value ∞ in the ratio game.

A Moore machine corresponding to an optimal strategy of Player 1 is robust and (strictly) realizes the error specification. If k is the value of the initial state then M is k-robust. ■

C. Synthesizing from Reset Error Specifications

As shown in Example 8 *reset error specifications* are an intuitive kind of error specification. We show here that every realizable reset error specification can be realized by a 1-robust Moore machine.

Definition 21. *A* reset error specification *is a double cost automaton with maximal cost* 1*, such that for all transitions (q,σ) with $c_e(q,\sigma) = 1$ or $c_s(q,\sigma) = 1$ the next state is $\delta(q,\sigma) = q_0$.*

Theorem 22. *Given a realizable reset error specification C, a* 1*-robust system can be synthesized in linear time.*

Proof: Translate C into a ratio game with a linear blowup, as in Lemma 19. We show that for an optimal strategy the ratio is not greater than 1. Let π_1 be a strategy such that for all resulting plays $\rho = q_0q_1\dots$, $\sum_{i=0}^{\infty} c_e(q_i, q_{i+1}) = 0$ implies $\sum_{i=0}^{\infty} c_s(q_i, q_{i+1}) = 0$. Thus, the system will not incur a cost from any state reachable using π_1 without environment cost. The only time a system cost may be incurred is when the environment incurs a cost of 1, in which case the system may also incur cost 1 and the system returns to the initial state. ■

VI. Related Work

We have defined a system to be robust if a small environment error leads to a small system error. Other approaches are possible. In the continuous domain, it is natural to require systems to be continuous, which guarantees robustness in the sense that a small output error can be guaranteed by an appropriately small input error [11]. This notion is not appropriate in the discrete setting, as discrete functions are in general not continuous. Consider, for example, a specification that requires that the value of the output g is always true (false) if the initial input r is true (false, respectively): $(r \rightarrow \mathsf{G}\, g) \wedge (\neg r \rightarrow \mathsf{G}\, \neg g)$. Here, a minimal difference in the input, namely a change of the initial input, causes a maximal difference in the output.

The importance of robustness is widely recognized. Rinard, for instance, advocates acceptability-oriented computing, stating that "complex computer systems should have a natural resilience to errors" [12].

Attie et. al [13] argue that fault-intolerant programs are often unrealistic. They introduce a framework to specify fault-tolerant concurrent programs with CTL formulas and differnent levels of tolerance, and show how to synthesize such programs. Contrary to our work, this work considers closed systems and requires the developer to specify possible faults explicitly. Cury and Krogh consider synthesis of robust controllers for discrete event systems, where a controller is optimal if it produces the correct behavior for a maximal set of plants including the original. This approach can beneficially be combined with ours to yield systems that fulfill the guarantees in a maximal set of cases and gracefully degrade otherwise.

Faella [14] considers the question of the appropriate behavior when a game is lost. He considers two notions, one based on dominating strategies and one based on a probability distribution over the input. In the former setting, he maximizes the set of inputs for which the game is won, and in the latter setting, the probability that the game is won. A similar problem is considered in [15], where an unrealizable specification G, which corresponds to a lost synthesis game, is generalized to a specification $A \rightarrow G$ for a maximally weak environment assumption A. None of these papers considers appropriate behavior in the cases where a system failure is inevitable, which is central to our notion of graceful degradation.

D'Souza and Gopinathan [16] consider a specification that is built from a ranked set of requirements, which may be contradictory. The requirements are "conflict tolerant", i.e., when overruled, they continue giving "advice." This is achieved through means closely related to our weighted edges. D'Souza and Gopinath describe how to synthesize controllers in which advice from a requirement is alternately followed and ignored. The question they answer is how to synthesize a system that always follows the highest ranked advice. The approach differs from ours in the focus on contradictory specifications rather than environment failures, and in the fact that the proper action is chosen greedily, whereas we solve a global optimization problem to find the appropriate behavior.

Alur, Kanade, and Weiss [17], consider prioritized requirements and present an efficient way to synthesize the highest priority requirement. This is related in the sense that the ideal specification may be left unfulfilled if necessary. What is missing, from our perspective, is a way to "return" to a higher-priority requirement.

Eisner considers properties in CTL of the form $\psi = \mathsf{AG}\, \phi$ (ϕ always holds) and calls a system robust if ψ holds in all states, not just in the reachable states. This implies that the system behaves well in the presence of environment failures (assuming that any invariants used as antecedents are weak), but Eisner states that control-intensive applications are typically not robust [18].

VII. Conclusions

We have introduced a notion of robustness for functional specifications based on graceful degradation. We have shown how to solve the verification problem and the synthesis problem for robust systems. The synthesis problem is solved through a novel type of games.

We consider the worst case only: when a specification only allows for k-robust systems, we do not distinguish between systems in which every trace is strictly k-robust and those in which some traces have fewer system faults. However, Chatterjee [personal communication] has shown that admissible (undominated) strategies do not always exist for mean-payoff games, and this result easily generalizes to ratio games, foiling the hope for a fully general solution. Another venue for improvement would be to minimize the constant d in the inequality between system and environment errors. Furthermore, it is an open question how to extend our approach to liveness.

It would be interesting to evaluate to which extent our notion of robustness matches the intuitive notions designers use.

References

[1] A. M. Davis, *Software Requirements — Analysis and Specification*. Prentice Hall, 1990.

[2] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of software engineering*. Prentice-Hall, Inc., 1991.

[3] A. Pnueli, "The temporal logic of programs," in *IEEE Symposium on Foundations of Computer Science*, Providence, RI, 1977, pp. 46–57.

[4] B. Jobstmann and R. Bloem, "Optimizations for LTL synthesis," in *FMCAD*, 2006, pp. 117–124.

[5] R. G. Schroeder, "Linear programming solutions to ratio games," *Operations Research*, 1970.

[6] H. Gimbert and W. Zielonka, "Games where you can play optimally without any memory," in *CONCUR*, 2005, pp. 428–442.

[7] U. Zwick and M. Paterson, "The complexity of mean payoff games on graphs," *Theoretical Computer Science*, vol. 158, pp. 343–359, 1996.

[8] E. Lawler, *Combinatorial Optimization: Networks and Matroids*. Courier Dover Publications, 1976.

[9] A. Dasdan, S. S. Irani, and R. K. Gupta, "Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems," in *DAC*, 1999, pp. 37–42.

[10] N. Piterman and A. Pnueli, "Faster solutions of Rabin and Streett games," in *LICS*, 2006, pp. 275–284.

[11] T. Henzinger, "Two challenges in embedded systems design: predictability and robustness," *Philosophical Trans. of the Royal Society*, 2008.

[12] M. C. Rinard, "Acceptability-oriented computing," in *OOPSLA Companion*, 2003, pp. 221–239.

[13] P. Attie, A. Arora, and E. A. Emerson, "Synthesis of fault-tolerant concurrent programs," *ACM Trans. Program. Lang. Syst.*, vol. 26, pp. 125–185, 2004.

[14] M. Faella, "Games you cannot win," in *Workshop on Games and Automata for Synthesis and Validation*, 2007.

[15] K. Chatterjee, T. Henzinger, and B. Jobstmann, "Environment assumptions for synthesis," in *CONCUR*, 2008, pp. 147–161.

[16] D. D'Souza and M. Gopinathan, "Conflict-tolerant features," in *CAV*, 2008, pp. 227–239.

[17] R. Alur, A. Kanade, and G. Weiss, "Ranking automata and games for prioritized requirements," in *CAV*, 2008, pp. 240–253.

[18] C. Eisner, "Using symbolic model checking to verify the railway stations of Hoorn-Kersenboogerd and Heerhugowaard," in *CHARME*, 1999, pp. 97–109.