Generalized Rabin(1) Synthesis with Applications to Robust System Synthesis*

Rüdiger Ehlers

Reactive Systems Group
Saarland University

Abstract. Synthesis of finite-state machines from linear-time temporal logic (LTL) formulas is an important formal specification debugging technique for reactive systems and can quickly generate prototype implementations for realizable specifications.

It has been observed, however, that automatically generated implementations typically do not share the robustness of manually constructed solutions with respect to assumption violations, i.e., they typically do not degenerate nicely when the assumptions in the specification are violated. As a remedy, robust synthesis methods have been proposed. Unfortunately, previous such techniques induced obstacles to their efficient implementation in practice and typically do not scale well.

In this paper, we introduce generalized Rabin(1) synthesis as a solution to this problem. Our approach inherits the good algorithmic properties of generalized reactivity(1) synthesis but extends it to also allow co-Büchi-type assumptions and guarantees, which makes it usable for the synthesis of robust systems.

1 Introduction

The problem of synthesizing finite-state systems from specifications written in linear-time temporal logic has recently received an increase in interest. Algorithmic advances in the solution of the synthesis problem have strengthened the practical applicability of synthesis algorithms and consequently, solution quality considerations that are common in the manual engineering process of reactive systems start to appear in the scope of synthesis as well.

In practice, many specifications consist of a set of assumptions the system to be synthesized can assume about the behavior of its environment, and a set of guarantees that it in turn has to fulfill. Such a situation is typical for cases in which a part of a larger system is to be synthesized. In this context, one particularly well-known solution quality criterion is the *robustness* of a system, i.e., how well it behaves under violations of the assumptions. As an example, a bus arbiter system could be designed to work in an environment in which not all clients request access to the bus as the same time. This assumption might however be violated if a part of the system breaks at runtime or errors were made in the engineering process. To counter these problems, manually constructed safety-critical systems are typically built in a way such that at least some

* An earlier version of this paper appeared as arXiv/CoRR document no. 1003.1684.

M. Bobaru et al. (Eds.): NFM 2011, LNCS 6617, pp. 101–115, 2011.

guarantees are still fulfilled in such a case. Automatically synthesized systems however typically do not exhibit robust behavior in such situations. As an example, the bus arbiter synthesized under this assumption could stop giving grants to clients completely once too many requests have occurred in a computation cycle.

To remedy these problems, a few techniques especially geared towards the synthesis of robust systems have been proposed. In [7], a robustness criterion and a synthesis algorithm based on cost automata have been defined, with the specification being restricted to consist of only safety properties. On the other hand, in [4], a robustness criterion based on the number of guarantees that still hold if assumptions are violated is defined, which connects robust synthesis to solving generalized Streett games. In both cases, the scalability of these techniques appears to be limited.

In this paper, we propose *generalized Rabin(1) synthesis*, an extension of the *generalized reactivity(1)* synthesis principle, originally proposed by Piterman, Pnueli and Sa'ar [20]. Our approach extends the expressivity of the latter approach while retaining its good algorithmic properties.

In particular, while generalized reactivity(1) synthesis is applicable to all specifications whose assumptions and guarantees are representable as deterministic Büchi automata, we extend its expressivity by allowing also one-pair Rabin-type and, as a special case, co-Büchi-type assumptions and guarantees, which are useful for representing *persistence* requirements [25]. Equally important, these extensions make the class of specifications that can be handled closed under applying a fairly straight-forward robustness criterion based on the number of computation cycles witnessing temporary violations of the assumptions and guarantees. At the same time, our approach inherits the good algorithmic properties of generalized reactivity(1) synthesis. Additionally, we show that any further non-trivial expressivity extension would result in losing these.

In the following, we describe two algorithms solving the robust synthesis problem. We start by showing how the generalized Rabin(1) synthesis problem can be reduced to solving a parity game with 5 colors and describe its use for robust system synthesis. Then, we discuss the fact that it is sometimes desirable to restrict the system to be synthesized to having some upper time bound between a temporary violation of a safety assumption and the final following violation of a safety guarantee afterwards, i.e., to return to normal operation after some upper time bound. For such cases, we present an adapted algorithm that has the additional advantage of extracting implementations having an extra output signal that reports whether the system is currently in the recovery mode after a violation of the assumption.

1.1 Related Work

Automatically synthesizing implementations from specifications given in linear-time temporal logic (LTL) is a well-studied problem in the literature. Its solutions can be classified into two sorts: (1) approaches that aim at handling the full expressivity of LTL, and (2) techniques that trade the full expressivity of LTL against algorithmic advantages. One particularly well-known approach of the latter kind, which we also build upon in this paper, is *generalized reactivity(1) synthesis* [20]. Its applicability in practice is witnessed by the existence of several successful case studies [5,6,17,25].

Synthesis of robust controllers is a relatively recent topic. In [3], Arora and Gouda describe the *closure* and *convergence* robustness criteria. Bloem et al. [4] solve the synthesis problem for the former criterion, where the number of guarantees that still hold in case of assumption violations is to be maximized. The corresponding synthesis problem is reduced to solving generalized Streett games. In [7], a quantitative approach for safety assumptions and guarantees is proposed, where the robustness definition is based on the number of computation cycles in which violations of the assumptions and guarantees are witnessed. This approach thus uses the convergence robustness criterion. Our work follows this line of research, but extends the idea to also allow liveness parts in the specification. The restriction to qualitative robustness improves the scalability of the proposed approach.

In the area of hybrid systems and control theory, work has been performed on robust synthesis where only the continuous part of the controller to be synthesized is to be made robust [25], while for the discrete part, generalized reactivity(1) synthesis is used. This paper can be seen as being orthogonal to that work as it solves the problem of introducing robustness in the discrete part of the controller. Thus, combining the two approaches leads to robustness of the overall solution synthesized. Also, in [25], so-called *stability* or *persistence* properties, which correspond to co-Büchi-type properties in the framework here, are used but applied to the generalized reactivity(1) synthesis approach in a way that leads to incompleteness of the overall procedure. Thus, our methods also increase the scope of properties expressible in that approach.

Closely related to robust synthesis is also the field of *fault-tolerant* synthesis. Here, fault models and fall-back specifications that need to hold in case of fault occurrences are explicitly given as input to the synthesis process. Most works in this area are concerned with adding robustness to completely specified systems, with a few exceptions (e.g., [11]). Our work follows the line of research in which the system to be synthesized should work in a reasonable way in case of assumption violations even if no explicit such fall-back specifications are given [7,4]. Thus, the techniques presented in this paper are even applicable if the fault model is unknown or it is not desired to invest time in writing the additional fall-back specifications.

2 Preliminaries

Words, Languages and natural numbers: Let Σ be a finite set. By Σ^*/Σ^ω we denote the set of all of its finite/infinite sequences, respectively. Such sequences are also called *words* over Σ. Sets of words are also called *languages*. For some sequence $w = w_0 w_1 \ldots$, we denote by w^j the suffix of w starting with the jth symbol, i.e., $w^j = w_j w_{j+1} \ldots$ for all $j \in \mathbb{N}$.

Mealy machines: *Reactive systems* are usually described using a *finite state machine* description. Formally, we define *Mealy machines* as five-tuples $\mathcal{M} = (S, \Sigma_I, \Sigma_O, \delta, s_0)$ where S is some finite set of states, Σ_I and Σ_O are input/output alphabets, respectively, $s_0 \in S$ is the initial state and $\delta : S \times \Sigma_I \to S \times \Sigma_O$ is the transition function of $\mathcal{M}$. The computation steps of a Mealy machine are called *cycles*.

For the scope of this paper, we set $\Sigma_I = 2^{\mathsf{AP}_I}$ and $\Sigma_O = 2^{\mathsf{AP}_O}$ for some sets of input/output atomic propositions AP_I and AP_O.

The languages induced by Mealy machines: Given a Mealy machine $\mathcal{M} = (S, \Sigma_I, \Sigma_O, \delta, s_0)$ and some input word $i = i_0 i_1 \ldots \in \Sigma_I^\omega$, $\mathcal{M}$ induces a *run* $\pi = \pi_0 \pi_1 \ldots$ and some *output word* $o = o_0 o_1 \ldots$ over i such that $\pi_0 = s_0$ and for all $j \in \mathbb{N}$: $\delta(\pi_j, i_j) = (\pi_{j+1}, o_j)$. Formally, we define the language of $\mathcal{M}$, written as $\mathcal{L}(\mathcal{M})$, to be the set of words $w = w_0 w_1 \ldots \in \Sigma^\omega$ with $\Sigma = 2^{\mathsf{AP}_I \uplus \mathsf{AP}_O}$ such that $\mathcal{M}$ induces a run π over the input word $i = w|_{\Sigma_I} = (w_0 \cap \Sigma_I)(w_1 \cap \Sigma_I) \ldots$ such that $w|_{\Sigma_O} = (w_0 \cap \Sigma_O)(w_1 \cap \Sigma_O) \ldots$ is the output word corresponding to π.

Linear-time temporal logic: For the description of the specification of a system, *linear-time temporal logic* (LTL) is a commonly used logic. Syntactically, LTL formulas are defined inductively as follows (over some set of atomic propositions AP):

- For all atomic propositions $x \in \mathsf{AP}$, x is an LTL formula.
- Let ϕ_1 and ϕ_2 be LTL formulas. Then $\neg\phi_1$, $(\phi_1 \vee \phi_2)$, $(\phi_1 \wedge \phi_2)$, $\mathsf{X}\phi_1$, $\mathsf{F}\phi_1$, $\mathsf{G}\phi_1$, and $(\phi_1 \mathsf{U} \phi_2)$ are also valid LTL formula.

The validity of an LTL formula ϕ over AP is defined inductively with respect to an infinite trace $w = w_0 w_1 \ldots \in (2^{\mathsf{AP}})^\omega$. Let ϕ_1 and ϕ_2 be LTL formulas. We set:

- $w \models p$ if and only if (iff) $p \in w_0$ for $p \in \mathsf{AP}$
- $w \models \neg\psi$ iff not $w \models \psi$
- $w \models (\phi_1 \vee \phi_2)$ iff $w \models \phi_1$ or $w \models \phi_2$
- $w \models (\phi_1 \wedge \phi_2)$ iff $w \models \phi_1$ and $w \models \phi_2$
- $w \models \mathsf{X}\phi_1$ iff $w^1 \models \phi_1$
- $w \models \mathsf{G}\phi_1$ iff for all $i \in \mathbb{N}$, $w^i \models \phi_1$
- $w \models \mathsf{F}\phi_1$ iff there exists some $i \in \mathbb{N}$ such that $w^i \models \phi_1$
- $w \models (\phi_1 \mathsf{U} \phi_2)$ iff there exists some $i \in \mathbb{N}$ such that for all $0 \leq j < i$, $w^j \models \phi_1$ and $w^i \models \phi_2$

We use the usual precedence rules for LTL formulas in order to be able to omit unnecessary braces and also allow the abbreviations typically used for Boolean logic, e.g., that $a \rightarrow b$ is equivalent to $\neg a \vee b$ for all formulas a, b.

Labeled parity games: A labeled parity game is a tuple $\mathcal{G} = (V_0, V_1, \Sigma_0, \Sigma_1, E_0, E_1, v_0, \mathcal{F})$ with V_0 and V_1 being the sets of vertices of the two players 0 and 1, Σ_0 and Σ_1 being their sets of actions, and $E_0 : V_0 \times \Sigma_0 \rightarrow V_1$ and $E_1 : V_1 \times \Sigma_1 \rightarrow V_0$ being their edge functions, respectively. We abbreviate $V = V_0 \uplus V_1$ and only consider finite games here, for which V_0, V_1, Σ_0 and Σ_1 are finite. The initial vertex v_0 is always a member of V_0. The coloring function $\mathcal{F} : V_0 \rightarrow \mathbb{N}$ assigns to each vertex in V_0 a color. For the scope of this paper, we only assign colors to vertices of player 0. We introduce the notation $\mathcal{F}^{-1}$ to denote the set of vertices of V_0 having a given color, i.e., for $c \in \mathbb{N}$, $\mathcal{F}^{-1}(c) = \{v \in V_0 : \mathcal{F}(v) = c\}$.

A decision sequence in $\mathcal{G}$ is a sequence $\rho = \rho_0^0 \rho_0^1 \rho_1^0 \rho_1^1 \ldots$ such that for all $i \in \mathbb{N}$, $\rho_i^0 \in \Sigma_0$ and $\rho_i^1 \in \Sigma_1$. A decision sequence ρ induces an infinite play $\pi = \pi_0^0 \pi_0^1 \pi_1^0 \pi_1^1 \ldots$ if $\pi_0^0 = v_0$ and for all $i \in \mathbb{N}$ and $p \in \{0, 1\}$, $E_p(\pi_i^p, \rho_i^p) = \pi_{i+p}^{1-p}$.

Given a play $\pi = \pi_0^0 \pi_0^1 \pi_1^0 \pi_1^1 \ldots$, we say that π is winning for player 0 if $\max\{\mathcal{F}(v) \mid v \in V_0, v \in \inf(\pi_0^0 \pi_1^0 \ldots)\}$ is even for the function $\inf$ mapping a sequence onto the set of elements that appear infinitely often in the sequence. If a play is not winning for player 0, it is winning for player 1.

Given some parity game $\mathcal{G} = (V_0, V_1, \Sigma_0, \Sigma_1, E_0, E_1, v_0, \mathcal{F})$, a strategy for player 0 is a function $f_0 : (\Sigma_0 \times \Sigma_1)^* \to \Sigma_0$. Likewise, a strategy for player 1 is a function $f_1 : (\Sigma_0 \times \Sigma_1)^* \times \Sigma_0 \to \Sigma_1$. In both cases, a strategy maps prefix decision sequences to an action to be chosen next. A decision sequence $\rho = \rho_0^0\rho_0^1\rho_1^0\rho_1^1 \dots$ is said to be in correspondence with f_p for some $p \in \{0, 1\}$ if for every $i \in \mathbb{N}$, we have $\rho_i^p = f_p(\rho_0^0\rho_0^1 \dots \rho_{i+p-1}^{1-p})$. A strategy is winning for player p if all plays in the game that are induced by some decision sequence that is in correspondence to f_p are winning for player p. It is a well-known fact that for parity games, there exists a winning strategy for precisely one of the players (see, e.g., [15]). We call a state $v \in V_0$ winning for player p if changing the initial state to v makes or leaves the game winning for player p. Likewise, a state $v' \in V_1$ is called winning for player p if a modified version of the game, that results from introducing a new initial state with only one transition to v' is (still) winning for player p.

If a strategy f_p for player p is a *positional strategy*, then $f_p(\rho_0^0\rho_0^1 \dots \rho_{n+p-1}^{1-p}) = f'_p(E_{1-p}\,(\dots E_1(E_0(v_0, \rho_0^0), \rho_0^1), \dots, \rho_{n+p-1}^{1-p}))$ for some function $f'_p : V_p \to \Sigma_p$. By abuse of notation, we call both f'_p and f_p positional strategies. Note that such a function f'_p is finitely representable as both domain and co-domain are finite. For parity games, it is known that there exists a winning positional strategy for a player if and only if there exists some winning strategy for the same player.

Note that a translation between this model and an alternative model where the coloring function is defined for both players is easily possible with only a slight alteration of the game structure.

ω-automata: An ω-automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, \mathcal{F})$ is a five-tuple consisting of some finite state set Q, some finite alphabet Σ, some initial state $q_0 \in Q$, some transition function $\delta : Q \times \Sigma \to 2^Q$ and some *acceptance component* $\mathcal{F}$ (to be defined later). We say that an automaton is deterministic if for every $q \in Q$ and $x \in \Sigma$, $|\delta(q, x)| \leq 1$. Given an ω-automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, \mathcal{F})$, we also call (Q, Σ, q_0, δ) the *transition structure* of $\mathcal{A}$.

Given an infinite word $w = w_0w_1 \dots \in \Sigma^\omega$ and an ω-automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, \mathcal{F})$, we say that some sequence $\pi = \pi_0\pi_1 \dots$ is a run for w if $\pi_0 = q_0$ and for all $i \in \mathbb{N}$, $\pi_{i+1} \in \delta(\pi_i, w_i)$. The language of $\mathcal{A}$, written as $\mathcal{L}(\mathcal{A})$, is the set of all words for which an accepting run through $\mathcal{A}$ exists. The acceptance of π by $\mathcal{F}$ is defined with respect to the type of $\mathcal{F}$, for which many have been proposed in the literature [15].

- For a *safety winning condition*, all infinite runs are accepting. In this case, the $\mathcal{F}$-symbol can also be omitted from the automaton definition.
- For a *Büchi acceptance condition* $\mathcal{F} \subseteq Q$, π is accepting if $\inf(\pi) \cap \mathcal{F} \neq \emptyset$. Here, $\mathcal{F}$ is also called the set of *accepting states*.
- For a *co-Büchi acceptance condition* $\mathcal{F} \subseteq Q$, π is accepting if $\inf(\pi) \cap \mathcal{F} = \emptyset$. Here, $\mathcal{F}$ is also called the set of *rejecting states*.
- For a *parity acceptance condition*, $\mathcal{F} : Q \to \mathbb{N}$ and π is accepting in the case that $\max\{\mathcal{F}(v) \mid v \in \inf(\pi)\}$ is even.
- For a *Rabin acceptance condition* $\mathcal{F} \subseteq 2^Q \times 2^Q$, π is accepting if for $\mathcal{F} = \{(E_1, F_1), \dots, (E_n, F_n)\}$, there exists some $1 \leq i \leq n$ such that $\inf(\pi) \cap E_i = \emptyset$ and $\inf(\pi) \cap F_i \neq \emptyset$.

– For a *Streett acceptance condition* $\mathcal{F} \subseteq 2^Q \times 2^Q$, π is accepting if for $\mathcal{F} = \{(E_1, G_1), \ldots, (E_n, F_n)\}$ and for all $1 \leq i \leq n$, we have $\inf(\pi) \cap E_i \neq \emptyset$ or $\inf(\pi) \cap F_i = \emptyset$.

For a one-pair Rabin automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, \{(E, F)\})$, we call F the *Büchi acceptance component* of the automaton while E is denoted as being the *co-Büchi acceptance component*. This terminology is justified by the fact that a one-pair Rabin automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, \{(E, F)\})$ accepts some word if and only if it is accepted by the co-Büchi automaton $\mathcal{A}_C = (Q, \Sigma, q_0, \delta, E)$ and the Büchi automaton $\mathcal{A}_B = (Q, \Sigma, q_0, \delta, F)$. Whenever a deterministic Rabin automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, (E, F))$ does not accept a word, we say that its *Büchi part is violated* if the states in F are visited only finitely often along the unique run, and say that its *co-Büchi part is violated* if some state in E is visited infinitely often along this run. Henceforth, we assume that all Büchi, co-Büchi, parity, Rabin and Streett automata are deterministic and without loss of generality, for all of their states $q \in Q$ and input symbols $x \in \Sigma$, we have $|\delta(q, x)| = 1$. We say that a parity automaton is weak if all states that are in the same strongly connected component have the same color.

Parity automata and parity games: Given a deterministic parity automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, \mathcal{F})$ with $\Sigma = 2^{(\mathsf{AP}_I \uplus \mathsf{AP}_O)}$, it is well-known that $\mathcal{A}$ can be converted to a parity game $\mathcal{G}$ such that $\mathcal{G}$ admits a winning strategy for player 1 (the so-called *system player*) if and only if there exists a Mealy machine $\mathcal{M}$ reading $\Sigma_I = 2^{\mathsf{AP}_I}$ and outputting $\Sigma_O = 2^{\mathsf{AP}_O}$ such that the language induced by $\mathcal{M}$ is a subset of the language of $\mathcal{A}$ (see, e.g., [23]). Furthermore, from a winning positional strategy in $\mathcal{G}$, such a Mealy machine $\mathcal{M}$ can easily be extracted.

Game solving and symbolic techniques: Many algorithms have been proposed for solving parity games, of which some are implementable symbolically, i.e., the sets of vertices and the edge functions can be represented implicitly by using, for example, binary decision diagrams (BDDs) [8,14,10]. In practice, BDDs have been shown to be useful when representing and computing properties of systems that are composed of many components that run in parallel [20,12,2]. One particularly important operation that needs to be performed in game solving is the computation of *attractor sets*. Given two sets of vertices A and B, we define $attr_p(A, B)$ to be the set of game vertices from which player p can enforce that eventually some vertex in B is visited while along the way, the set of vertices A is not left. For the scope of this paper, we let $attr_p$ deal only with vertices of player 0, i.e., A and B may only contain vertices of player 0 and we do not restrict visits to vertices of player 1. The attractor set and a corresponding strategy for player p can be computed symbolically [2,1].

Specifications: In this paper, we consider specifications of the form $\psi = (a_1 \wedge a_2 \wedge \ldots \wedge a_{n_a}) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_{n_g})$. By abuse of notation, we allow both LTL formulas and deterministic automata as *assumptions* $\{a_1, \ldots, a_{n_a}\}$ and *guarantees* $\{g_1, \ldots, g_{n_g}\}$. A word $w \in (2^{\mathsf{AP}})^\omega$ satisfies ψ if either for some LTL assumption a_i, $w \not\models \psi$, for some assumption automaton a_i, $w \notin \mathcal{L}(a_i)$, or for all LTL and automata guarantees g_i, $w \models g_i$ and $w \in \mathcal{L}(g_i)$, respectively. We assume that all LTL formulas in ψ range over the same set of atomic propositions AP and all automata use 2^{AP} as alphabet. Converting an LTL formula to an equivalent deterministic automaton is a classical topic in the literature, is

explained in [15] and nowadays, suitable tools are available [16]. We say that an LTL formula has a Rabin index of one if it can be converted to Rabin automata with one acceptance pair.

In this paper, we are especially interested in specifications in which the assumption and guarantee conjuncts are of the forms ψ, $\mathsf{G}\psi$, $\mathsf{GF}\psi$, $\mathsf{F}\psi$ and $\mathsf{FG}\psi$ with the only LTL temporal operator occurring in ψ being X. These are called *initialization*, *basic safety*, *basic liveness*, *eventuality* and *persistence* properties.

3 Generalized Rabin(1) Synthesis

In this section, we present the core construction of the generalized Rabin(1) synthesis approach (abbreviated by GRabin(1) in the following) and then prove that its scope cannot be extended without losing its good properties. Afterwards, we discuss its application to the synthesis of robust systems. We start with a specification of the form

$$\psi = (a_1 \wedge a_2 \wedge \ldots \wedge a_{n_a}) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_{n_g})$$

for some set of assumptions $\{a_1, \ldots, a_{n_a}\}$ and some set of guarantees $\{g_1, \ldots, g_{n_g}\}$. We assume that these are given in form of deterministic one-pair Rabin automata. Most specifications found in practice can be converted to such a form using commonly known techniques [21,18,16].

Our construction transforms such a specification to a deterministic parity automaton with at most 5 colors that accepts precisely the words that satisfy ψ. The number of states of the generated automaton is polynomial in the product of the state numbers of the individual Rabin automata $a_1, \ldots, a_{n_a}, g_1, \ldots, g_{n_g}$. The generated parity automaton can then be syntactically transformed into a parity game (taking into account the partitioning of the atomic propositions into input and output bits) that is winning for player 1 if and only if there exists a Mealy machine over the given sets of inputs and outputs such that all of its runs satisfy the specification. By using for example the parity game solving algorithm by McNaughton/Zielonka [19], the realizability problem is then solvable symbolically. This algorithm is constructive, i.e., it is able to produce a winning strategy that can be used as a prototype implementation.

Let A be the set of assumption one-pair Rabin automata and G be the set of such guarantee automata. For improved readability of the following description of the algorithm, by abuse of notation, we introduce δ, Q, q_0, Σ, and $\mathcal{F}$ as functions mapping automata onto their components. For example, given some automaton $\mathcal{A} = (\tilde{Q}, \tilde{\Sigma}, \tilde{q}_0, \tilde{\delta}, (\tilde{E}, \tilde{F}))$, we have $\delta(\mathcal{A}) = \tilde{\delta}$.

For $A = \{a_1, \ldots, a_{n_a}\}$ and $G = \{g_1, \ldots, g_{n_g}\}$, we construct the deterministic parity automaton $\mathcal{A}' = (Q', \Sigma', \delta', q'_0, \mathcal{F}')$ that accepts precisely the words on which ψ is satisfied as follows:

- Σ' is chosen such that for all $a \in A \uplus G$: $\Sigma' = \Sigma(a)$
- $Q' = Q(a_1) \times \ldots \times Q(g_{n_g}) \times \{0, 1, \ldots, n_a\} \times \{0, 1, \ldots, n_g\} \times \mathbb{B}$
- For all $q = (q^a_1, \ldots, q^g_{n_g}, q^W, q^R, q^V) \in Q'$ and $x \in \Sigma'$, we define $\delta'(q, x) = (q'^a_1, \ldots, q'^g_{n_g}, q'^W, q'^R, q'^V)$ such that:
  - For all $1 \leq i \leq n_a$: $\delta(a_i)(q^a_i, x) = q'^a_i$

  - For all $1 \leq i \leq n_g$: $\delta(g_i)(q_i^g, x) = q_i'^g$
  - $q'^W = (q^W + 1) \mod (n_a + 1)$ if $q'^a_{q^W} \in F(a_{q^W})$ or $q^W = 0$, otherwise $q'^W = q^W$.
  - $q'^R = (q^R + 1) \mod (n_g + 1)$ if $q'^g_{q^R} \in F(g_{q^R})$ or $q^R = 0$, otherwise $q'^R = q^R$.
  - $q'^V = \mathbf{true}$ if and only if (at least) one the following two conditions hold:
    - $q^W = 0$
    - for all $1 \leq i \leq n_g$, $q_i'^g \notin E(g_i)$ and $q^V = \mathbf{true}$
- For all $q = (q_1^a, \ldots, q_{n_g}^g, q^W, q^R, q^V) \in Q'$, we have that $\mathcal{F}'$ maps q to the least value in $c \in \{0, 1, 2, 3, 4\}$ such that:
  - $c = 4$ if for some $1 \leq i \leq n_a$: $q_i^a \in E(a_i)$
  - $c \geq 3$ if $q^V = \mathbf{true}$ and for some $1 \leq i \leq n_g$, $q_i^g \in E(g_i)$
  - $c \geq 2$ if $q^R = 0$.
  - $c \geq 1$ if $q^W = 0$
- $q_0' = (q_0(a_1), \ldots, q_0(g_{n_g}), 0, 0, \mathbf{false})$

The components $q_1^a, \ldots, q_{n_g}^g$ in a state tuple $q = (q_1^a, \ldots, q_{n_g}^g, q^W, q^R, q^V) \in Q'$ represent the automata of $A \uplus G$ running in parallel. The remaining part of the state tuples corresponds to some additional *control structure* for checking if the overall specification is satisfied. Note that adding the control structure only results in a polynomial blow-up. The parts of the control structure have the following purposes:

- The counter q^W keeps track of the assumption automaton number for which an accepting state in its Büchi component is to be visited next. The construction is essentially the same as for de-generalizing generalized Büchi automata (see, e.g., [22]).
- The counter q^R does the same for the guarantees.
- The bit q^V tracks if accepting states for the Büchi components of all automata in A have been visited since the last visit to a rejecting state for the co-Büchi component of some guarantee.

A full proof of the correctness of the construction can be found in [13].

3.1 Extending Generalized Rabin(1) Synthesis

The generalized Rabin(1) synthesis approach presented above is capable of handling all assumptions and guarantees that have a Rabin index of one. A natural question to ask at this point is whether the approach can be extended in order to be also able to handle specifications with conjuncts of a higher Rabin index without losing its good properties. These are:

- the fact that the state space of the generated parity automaton is the product of the state spaces of the individual automata and some polynomially sized control structure, which makes the automaton state space amenable to an encoding using symbolic techniques, and
- the constant number of colors, which allows the application of efficient symbolic parity game solving algorithms.

Unfortunately, the approach cannot be extended while retaining these advantages. To see this, consider Streett game solving, which is known to be co-NP-complete [15]. If we were able to accommodate one-pair Streett automata (which are a special case of two-pair Rabin automata) as guarantees in the synthesis approach presented, we could decompose a Streett automaton with n acceptance pairs (for some $n \in \mathbb{N}$) into n one-pair Streett automata, each having the transition structure of the original Streett automaton, take these as guarantees and use no assumptions. The specification is then realizable if and only if it is realizable for the original Streett automaton. Since however, the individual one-pair Streett automata have the same transition structure and thus transition in a synchronized manner, if we were able to build a parity game having the properties stated above, the parity game would only have a number of vertices polynomial in the size of the original Streett automaton and thus, due to the constant number of colors, the realizability problem for the original Streett automaton would be solvable in polynomial time. So the existence of a similar algorithm for *generalized Streett(1) synthesis* or *generalized Rabin(2) synthesis* would imply P=NP.

3.2 Application to Synthesize Robust Systems

Assume that we have a specification of the form $(a_1 \wedge a_2 \wedge \ldots \wedge a_{n_a}) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_{n_g})$, where all assumptions and guarantees are either initialization, basic safety, basic liveness or persistence properties. During the run of a system satisfying ψ, the assumptions may be violated temporarily. A common criterion for the robustness of a system is that in such a case, it must at some point return to *normal operation mode* after such a temporary assumption violation [7,3]. In the scope of synthesis, implementing such a convergence [3] criterion requires fixing a definition of temporary assumption violations. Taking a specification of the form stated above, only a violation of the initialization or basic safety assumptions can be detected during the run of the system. Moreover, only the basic safety properties can be violated temporarily as an initialization property is only evaluated at the start of a system run. Thus we define:

Definition 1. *Given a word $w = w_0 w_1 \ldots \in (2^{\mathsf{AP}})^\omega$ and an LTL formula $\psi = \mathsf{G}\phi$, we say that position $i \in \mathbb{N}$ in the word witnesses the non-satisfaction of ψ if there exists some $j \leq i$ such that for no $w' \in (2^{\mathsf{AP}})^\omega$, $w_j \ldots w_i w' \models \phi$. Furthermore, given a specification of the form $(a_1 \wedge a_2 \wedge \ldots \wedge a_{n_a}) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_{n_g})$, where all assumptions are initialization, basic safety, basic liveness or persistence properties, we say that the assumptions/guarantees are temporarily violated on a word w at position i if position i witnesses the non-satisfaction of some basic safety assumption/guarantee in the specification, respectively.*

In [3], convergence has been defined for the safety case. In this paper, we extend the definition to the liveness and persistence cases:

Definition 2. *Given a specification of the form $(a_1 \wedge a_2 \wedge \ldots \wedge a_{n_a}) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_{n_g})$, where all assumptions and guarantees are initialization, basic safety, basic liveness or persistence properties, we say that a system converges if the following conditions hold for all words in the language of the system:*

- *there exists a bound on the number of temporary basic safety guarantee violations in between any two temporary basic safety assumption violations and after the last temporary basic safety assumption violation, and*
- *If w is a word in the language of the system satisfying the initialization assumptions and for some $j \in \mathbb{N}$, w^j satisfies all non-initialization assumptions, then w satisfies the initialization guarantees and for some $j' \geq j$, $w^{j'}$ satisfies all non-initialization guarantees.*

In this definition, there is no requirement that a converging system also performs some progress on its liveness (and persistence) properties in between two temporary assumption violations even if they are sufficiently sparse, which in practice a robust system should surely do. Nevertheless, we argue that for the scope of synthesis, this definition is still useful. The reason is that all synthesis procedures used nowadays produce finite-state solutions. Thus, if temporary assumption violations stop occurring for a couple of computation cycles and at the same time, some progress is made with respect to liveness assumptions (i.e., accepting states are visited for their automata), the system has, after a finite period of time, also continue to work towards fulfilling the liveness guarantees. If this was not the case, we could find a loop in the finite-state machine description of the system that witnesses non-convergence, which would be a contradiction. Given that our synthesis procedure only produces finite-state solutions (as parity game solving algorithms typically do), it is thus enough to require that after the *last* temporary assumption violation the system converges in order to ensure that the system converges in general. We can easily express convergence after the last temporary assumption violation in LTL by prefixing the guarantees and the basic safety assumptions in the specification using the F (finally) operator of LTL.

Definition 3. *Given a specification of the form $\psi = (a_1 \wedge a_2 \wedge \ldots \wedge a_{n_a}) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_{n_g})$, where all assumptions and guarantees are initialization, basic safety, basic liveness or persistence properties, and $a_s, \ldots, a_{n_a}$ are precisely the basic safety assumptions, we define the ruggedized version of ψ to be:*

$$\psi' = (a_1 \wedge \ldots \wedge a_{s-1} \wedge \mathsf{F}(a_s) \wedge \ldots \mathsf{F}(a_{n_a})) \rightarrow (\mathsf{F}(g_1) \wedge \mathsf{F}(g_2) \wedge \ldots \wedge \mathsf{F}(g_{n_g}))$$

Note that when taking a generalized Rabin(1) specification consisting only of initialization, basic safety, basic liveness and persistence properties, ruggedizing it does not change its membership in this class, as basic safety properties are converted to persistence properties, basic liveness properties stay untouched (as $\mathsf{FGF}(\phi)$ is equivalent to $\mathsf{GF}(\phi)$ for all LTL formulas ϕ) and likewise, persistence properties are not altered. This property does not hold for generalized reactivity(1) specifications, as the ruggedization process converts pure safety properties to persistence properties.

Thus, we can solve the robust synthesis problem, where converging systems are to be found that satisfy a given specification, by ruggedizing the specification and using the generalized Rabin(1) synthesis approach. The convergence criterion above however does not state that under no safety assumption violation, also no guarantee violation should be performed by the system to be synthesized, which is also required in the vast majority of practical cases where synthesis can be applied. In order to incorporate this requirement to the synthesis process, we can build a deterministic weak automaton

from the original specification but using only safety assumptions and guarantees. By taking the conjunction of this automaton with the parity automaton obtained from the ruggedized specification using the construction stated above, we obtain a five-color parity game for which all implementations realizing the specification also have this additional property.

So far, we have required all assumption and guarantee conjuncts to be initialization, basic safety, basic liveness and persistence properties. We leave the question how to ruggedize specifications consisting of arbitrary one-pair Rabin automata as assumptions and guarantees open, as there is no extension to the ruggedization concept that is suitable in general. As an example, in general we could have the assumption in a specification that at precisely every second computation cycle, some input bit should be set to $\mathbf{true}$. If during the execution of the system, the environment flips the phase of the signal (e.g., from setting the bit to $\mathbf{true}$ every even cycle to setting it to $\mathbf{true}$ every odd cycle), whether this should count as only a temporary violation or a permanent one depends on the application. Thus, a generic ruggedization construction for specifications that do not only have initialization, basic safety or liveness and persistence conjuncts cannot be given.

4 Bounded-Transition-Phase Generalized Rabin(1) Synthesis

In the preceding section, we defined generalized Rabin(1) synthesis and its application to synthesize robust systems. These systems have the property that after a temporary violation of some assumption, the system returns to *normal operation mode* after a finite period of time. The system might however have the drawback that the length of the period is not under its control and might grow arbitrarily. Consider the following example, which is a realizable specification over $\mathsf{AP}_I = \{i\}$ and $\mathsf{AP}_O = \{o\}$:

$$(i \wedge \mathsf{GF}i \wedge \mathsf{G}(i \leftrightarrow \mathsf{X}i)) \rightarrow (\mathsf{G}(o \leftrightarrow i) \wedge \mathsf{G}o)$$

The environment can temporarily violate its specification by switching from continuously choosing $i = \mathbf{true}$ to $i = \mathbf{false}$ and vice versa. While the environment plays a stream of $i = \mathbf{false}$, the system has to violate some of its guarantees. However, the environment has to switch back to setting $i = \mathbf{true}$ at some point in order not to violate its liveness property. Thus, also the ruggedized version of the specification is realizable. However, we cannot give a time bound on the duration of a phase in which i is set to $\mathbf{false}$ continuously and thus, there is no implementation of the specification that has a time bound on the length of the *transition phase* in which the system switches back to normal operation mode after the last temporary assumption violation.

Definition 4. *Given a specification $\psi = (a_1 \wedge a_2 \wedge \ldots \wedge a_{n_a}) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_{n_g})$, where all assumptions and guarantees are initialization, basic safety, basic liveness or persistence properties, we say that a robust system is in normal operation mode with respect to ψ after the input/output prefix word $w \in (2^{\mathsf{AP}_I \uplus \mathsf{AP}_O})^*$ if the system can enforce that the postfix word $w' \in (2^{\mathsf{AP}_I \uplus \mathsf{AP}_O})^\omega$ representing the following input/output either has the property that the initialization assumptions are not satisfied in ww', or no safety guarantee temporary violation is witnessed in ww' from position $|w|+1$ onwards before the next safety assumption violation.*

We say that a system is in *recovery mode* whenever it is not in normal operation mode. Undoubtedly, systems that guarantee an upper time bound on the number of computation cycles being in recovery mode after a temporary assumption violation has occurred (provided that no further such violation occurs during the recovery process) are more desirable [9]. In this section, we show how to obtain these with the generalized Rabin(1) synthesis approach, whenever possible. As a side-result, the systems synthesized also have an additional output bit that always indicates whether normal operation mode has already been restored.

We solve the *bounded-transition-phase* generalized Rabin(1) synthesis problem by borrowing ideas from [9], where finitary winning conditions for parity games are introduced, but only apply these to the co-Büchi part of the specification that has been introduced when ruggedizing the original specification. In contrast to [9], we thus avoid that specifications in which the system to be synthesized is required to wait for some external events for fulfilling its obligations become unrealizable, which applies to the majority of the industrial case studies available for generalized reactivity(1) synthesis at the moment (see, e.g., [5,6,26]).

Let ψ be a generalized reactivity(1) specification. We start by ruggedizing ψ and convert the resulting LTL formula to a deterministic parity automaton $\mathcal{A}'$ as described in the previous section, but this time modify the construction slightly to always set the q^V flag to $\mathbf{true}$. With this modification, if some rejecting state for the co-Büchi-part of some guarantee Rabin automaton is visited along a run, this results in an occurrence of color 3, except if at the same time some state in the co-Büchi-part of an assumption is visited (which leads to color 4). When computing the automata from the specification conjuncts, for the persistence properties, we make sure that their automata are *co-Büchi-tight*. We say that a deterministic co-Büchi word automaton is co-Büchi-tight for some LTL formula $\psi = \mathsf{FG}\phi$ if a run of the automaton visits a rejecting state precisely at the positions in the corresponding word that witness the non-satisfaction of $\mathsf{G}\phi$. We define:

Definition 5. *Given a parity game $\mathcal{A}$ with colors $\{0,1,2,3,4\}$, we say that some strategy f for player 1 is a winning **color-3 bounded-transition-phase strategy** if there exists some constant $c \in \mathbb{N}$ such that on every run of $\mathcal{A}$ that is in correspondence to f, a visit to color 3 along the run can only occur within c steps after an occurrence of color 4.*

Whenever we have a winning color-3 bounded-transition-phase strategy for the game induced by $\mathcal{A}'$, the strategy represents an implementation realizing ψ using only a bounded transition-phase before returning to normal operation mode after a temporary assumption violation, as in $\mathcal{A}'$, all visits to color 3 signal visits to co-Büchi states in some Rabin guarantee automaton, which in turn witness temporary guarantee violations. Thus, such a strategy represents a system implementation for which the number of computation steps in which it can be in recovery mode between any two temporary assumption violations is limited.

Let $\mathcal{G} = (V_0, V_1, \Sigma_0, \Sigma_1, E_0, E_1, v_0, c)$ be the game built from $\mathcal{A}'$. We use a function $parity_p(B, C)$ for computing the set of winning vertices in a parity game, i.e., for $p \in \{0,1\}$, it maps the sets of V_0-vertices B and C onto the set of vertices from which player p can win the game, assuming all states in B to be winning for this player and all states in C to be winning for the other player. We assume that the $parity_p$ function

also computes a winning strategy. Note that the commonly used parity game solving algorithms can easily be modified to handle such additional parameter sets B and C and to compute such a strategy [14,10].

At any point during a run in which player 1 plays a bounded-transition strategy, she is either in the transition phase or in a vertex from which she can win without being in the transition phase, i.e., from which she has a winning strategy that does not visit a vertex with color 3 before a vertex with color 4 is visited and at the same time, if a color-4-vertex is never visited again, is winning using only vertices with colors 0, 1 and 2. Using this observation, we can compute the set of V_0-vertices from which player 1 can win while being in the transition phase using a fixed point characterization of the set:

$$W = \nu X.X \wedge attr_1(\mathcal{F}^{-1}(4) \vee parity_1(\mathcal{F}^{-1}(4) \wedge X, \mathcal{F}^{-1}(3))) \tag{1}$$

The subformula $parity_1(\mathcal{F}^{-1}(4) \wedge X, \mathcal{F}^{-1}(3))$ computes from which vertices the game can be won when not being in the transition phase, assuming that the states in X are winning during the transition phase, as a visit to color 4 allows switching to the transition phase, and a vertex with color 3 must not be visited beforehand. Taking the attractor set of $parity_1(\mathcal{F}^{-1}(4) \wedge X, \mathcal{F}^{-1}(3))$ allows finding the vertices from which player 1 can enforce that either a vertex with color 4 is visited after a finite period of time or that in a finite number of steps, vertices are reached which are winning even when not being in the transition phase. Using this set W, the set of vertices from which player 1 can win when not being in the transition phase can then be obtained by computing:

$$Y := parity_1(\mathcal{F}^{-1}(4) \wedge W, \mathcal{F}^{-1}(3) \wedge W) \tag{2}$$

Theorem 1. *Given a parity game $\mathcal{G}$, the set of the vertices of player 0 in a game from which a winning color-3 bounded-transition-phase strategy exists for player 1 is equal to the set of vertices computed by Equation 2.*

Note that in this setting, the *parity* function only needs to deal with three-color parity games, as the vertices with color 4 and 3 are assumed to be winning and losing for the system player, respectively, so they can be remapped to color 0 for the scope of the *parity* function. This allows the usage of specialized three-color parity game solving algorithms such as the one described in [10], which has been shown to work well for symbolic game solving in practice. Also, as the computations of attractors, conjunctions, disjunctions and fixed points can be done symbolically [14], we obtain a fully symbolic algorithm for computing the states from which a winning color-3 bounded-transition-phase strategy exists.

Extracting such a strategy is also simple. While being in Y, the system player plays the strategy computed by the *parity* function. Whenever this is not the case, it moves along the attractor towards Y. Since an implementation can track its current vertex in the game, it can easily output a signal stating whether it is in Y or not. This signal then serves as an indicator whether the system is in normal operation or recovery mode.

The description of the algorithm established in this section does not immediately generalize to the case that the original specification also contains co-Büchi objectives, as co-Büchi rejecting states may be visited before the first occurrence of a safety assumption violation in practice, but after the ruggedization of the specification and the

conversion to a parity game, such a visit is not allowed by a color-3 bounded-transition-phase strategy. It is however easy to adapt our algorithm to fix this problem: we additionally introduce the colors 5 and 6 and map violations of the co-Büchi parts of Rabin guarantee and assumption automata that correspond to safety conjuncts in the original specification to these colors. The colors 3 and 4 are then still used for the persistence properties of the original specification, using the q^V bit of the construction from Sect. 3. Then, we only need to search for color-5 bounded-transition-phase strategies instead of color-3 ones and alter the equations 1 and 2 to $W = \nu X.X \wedge attr_1(\mathcal{F}^{-1}(5) \vee parity_1(\mathcal{F}^{-1}(6) \wedge X, \mathcal{F}^{-1}(5)))$ and $Y := parity_1(\mathcal{F}^{-1}(6) \wedge W, \mathcal{F}^{-1}(5))$.

5 Conclusion

In this paper, we presented generalized Rabin(1) synthesis. We have shown that our approach is the maximally possible extension to generalized reactivity(1) synthesis that has the same good algorithmic properties. As an application, we have defined a robustness criterion suitable for specifications consisting of *initialization*, *basic safety*, *basic liveness*, and *persistence* conjuncts and shown that the set of generalized Rabin(1) specifications consisting only of these conjuncts is closed under the process of ruggedization, which automatically transforms a specification into one for a system that needs to be robust against environment assumption violations. By applying a special algorithm for bounded-transition-phase generalized Rabin(1) synthesis, we can furthermore search for implementations that are even more robust in the sense that the transition phase between the normal operation mode and the recovery mode after a temporary assumption violation has to be bounded in length by some constant.

The practical applicability of the techniques described in this paper is witnessed by the fact that in robotics, where generalized reactivity(1) synthesis starts to be applied, the inability of generalized reactivity(1) synthesis to handle co-Büchi-type specifications is discussed in some publications in that field (see, e.g., [17,26]). Our techniques allow specifying such properties and are implementable in a fully symbolic manner. When applied to only Büchi-type assumptions and guarantees, the basic GRabin(1) synthesis approach is equivalent to the one for generalized reactivity(1) synthesis described in [4] and as a lightweight modification to the former algorithm, it inherits its good algorithmic properties.

Acknowledgements. This work was supported by the German Research Foundation (DFG) within the program "Performance Guarantees for Computer Systems" and the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

The author wants to thank Roderick Bloem and Krishnendu Chatterjee for interesting discussions and ideas on [13].

References

1. Alur, R., Henzinger, T.A., Mang, F.Y.C., Qadeer, S., Rajamani, S.K., Tasiran, S.: Mocha: Modularity in model checking. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 521–525. Springer, Heidelberg (1998)

2. Alur, R., Madhusudan, P., Nam, W.: Symbolic computational techniques for solving games. STTT 7(2), 118–128 (2005)
3. Arora, A., Gouda, M.G.: Closure and convergence: A foundation of fault-tolerant computing. IEEE Trans. Software Eng. 19(11), 1015–1027 (1993)
4. Bloem, R., Chatterjee, K., Greimel, K., Henzinger, T.A., Jobstmann, B.: Robustness in the presence of liveness. In: [24], pp. 410–424
5. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Interactive presentation: Automatic hardware synthesis from specifications: a case study. In: Lauwereins, R., Madsen, J. (eds.) DATE, pp. 1188–1193. ACM, New York (2007)
6. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Specify, compile, run: Hardware from PSL. Electr. Notes Theor. Comput. Sci. 190(4), 3–16 (2007)
7. Bloem, R., Greimel, K., Henzinger, T.A., Jobstmann, B.: Synthesizing robust systems. In: FMCAD, pp. 85–92. IEEE, Los Alamitos (2009)
8. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)
9. Chatterjee, K., Henzinger, T.A., Horn, F.: Finitary winning in omega-regular games. ACM Trans. Comput. Log. 11(1) (2009)
10. de Alfaro, L., Faella, M.: An accelerated algorithm 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)
11. Dimitrova, R., Finkbeiner, B.: Synthesis of Fault-Tolerant Distributed Systems. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 321–336. Springer, Heidelberg (2009)
12. Ehlers, R.: Symbolic bounded synthesis. In: [24], pp. 365–379
13. Ehlers, R.: Generalised Rabin(1) synthesis. arXiv/CoRR abs/1003.1684 (2010)
14. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy (extended abstract). In: FOCS, pp. 368–377. IEEE, Los Alamitos (1991)
15. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research. LNCS, vol. 2500. Springer, Heidelberg (2002)
16. Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theor. Comput. Sci. 363(2), 182–195 (2006)
17. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE Transactions on Robotics 25(6), 1370–1381 (2009)
18. Krishnan, S.C., Puri, A., Brayton, R.K., Varaiya, P.: The Rabin index and chain automata, with applications to automatas and games. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 253–266. Springer, Heidelberg (1995)
19. McNaughton, R.: Infinite games played on finite graphs. Ann. Pure Appl. Logic 65(2), 149–184 (1993)
20. Piterman, N., Pnueli, A., Sa'ar, 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)
21. Safra, S.: Complexity of Automata on Infinite Objects. PhD thesis, Weizmann Institute of Science, Rehovot, Israel (March 1989)
22. Thomas, W.: Automata on Infinite Objects. In: Handbook of Theoretical Computer Science. Formal Models and Semantics, vol. B, pp. 133–191. MIT Press, Cambridge (1994)
23. Thomas, W.: Church's problem and a tour through automata theory. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 635–655. Springer, Heidelberg (2008)
24. Touili, T., Cook, B., Jackson, P. (eds.): CAV 2010. LNCS, vol. 6174. Springer, Heidelberg (2010)
25. Wongpiromsarn, T., Topcu, U., Murray, R.M.: Automatic synthesis of robust embedded control software. In: AAAI Spring Symposium on Embedded Reasoning (2010)
26. Wongpiromsarn, T., Topcu, U., Murray, R.M.: Receding horizon control for temporal logic specifications. In: Johansson, K.H., Yi, W. (eds.) HSCC, pp. 101–110. ACM, New York (2010)