Synthesizing Reactive Systems Using Robustness and Recovery Specifications

Roderick Bloem[1] Hana Chockler[2] Masoud Ebrahimi[1] Ofer Strichman[3]
[1]Graz University of Technology [2]King's College London [3]Information Systems Engineering, IE, Technion

***Abstract*—Past literature on synthesis identified the need to synthesize systems that are *robust* to failures of the system in reading the inputs from the environment, and also to failures of the environment itself to satisfy our assumptions about its behavior. In this work, we propose a simple and flexible framework for synthesizing robust systems, where the user defines the required robustness via a temporal *robustness specification*. For example, the user may specify that the environment is *eventually* reliable, or input misreadings cannot occur more than k consecutive steps, and synthesize a system under this assumption. Furthermore, our framework enables us to specify, also, a temporal *recovery specification*, i.e., describing the way the system is expected to recover after a failure of the environment assumptions. We show examples of robust systems that we have synthesized with this method by our synthesis tool PARTY.**

I. INTRODUCTION

Given a temporal property φ_s, the problem of *reactive synthesis* is to generate a system M such that $M \models \varphi_s$. When computationally possible it may save labor in comparison to the more traditional route of manually constructing M and then verifying that it satisfies φ_s via model-checking.

It is common to synthesize a system for a *given* environment rather than requiring that it operates correctly under a completely *free* environment, i. e., under arbitrary sequences of inputs. Thus, our assumptions about the environment's behavior is part of the input to the synthesis algorithm. Specifically, we require

$$M \models \varphi_e \rightarrow \varphi_s \ , \tag{1}$$

where φ_e is a (temporal) formula specifying the environment's expected behavior.

In practice, it is very hard to fully specify a system, i.e., specify the exact behavior in reaction to any sequence of inputs from the environment. As a result, there can be many such systems M that satisfy φ_s, and hence it is desirable to define default criteria for the quality of the synthesized system, e.g., [1–3], which can guide us in the synthesis process. In this sense, it is like an optimization problem without an explicit objective function. One such quality measure is *robustness*, a term that was used with multiple meanings in, e.g., [4–8]. Informally, robustness measures the degree to which a system can function correctly in the presence of erroneous input or stressful environmental conditions [9]. Indeed it is common that the behavior of the environment under which M is expected to operate is not fully specified, cannot be modeled in a precise way (e. g., if it is a physical environment), or cannot be fully trusted. Hence when possible, M should satisfy φ_s even when φ_e does not hold, or when some inputs are misread by the system or misreported by the environment. These extra requirements may depend on the type of system and the expected environment in which it operates, and accordingly on the selected definition of robustness.

The main contribution of this article is a unified method of reducing the problem of robust synthesis, for almost any conceivable definition of robustness, to the problem of 'normal' synthesis. This stands in stark contrast to the above-mentioned prior works on robustness, each of which offered its specialized algorithm to cope with this problem. The reduction is simple, and amounts to writing a *robustness specification* in LTL. We further offer another level of flexibility in the framework, by allowing the user to write a *recovery specification* in LTL, which defines how the system recovers from failures. Our method can be used with any off-the-shelf synthesizer that supports synthesis with incomplete information [10], by simply modifying the specification. We implemented an even easier route in PARTY [11] (via bounded synthesis) where in addition to φ_e, φ_s, only the robustness and recovery specifications have to be added.

II. DEFINITIONS OF ROBUSTNESS

We will distinguish between two classes of robustness definitions. The first is robustness against 'hidden' inputs, which means that the system fails to read the environment's signals; The second is robustness against cases in which the environment does not satisfy the assumption φ_e.

Let us now survey various definitions of robustness in these two classes. Those that are described below without a reference are novel to this work.

Definitions of robustness against hidden inputs:

- **(R1)** A robust system satisfies φ_s even if some of the inputs can be misread ('hidden') in a finite number of steps.
- **(R2)** k-robustness (inputs): A robust system satisfies φ_s even if it misreads some of the inputs up to k times consecutively.
- **(R3)** A system is robust if it satisfies φ_s even if some or all the inputs are occasionally hidden.

Definitions of robustness against violations of the assumptions:

- **(R4)** A system is robust if despite a finite number of violations of the assumptions, it still satisfies the guarantees φ_s.
- **(R5)** According to [5] a robust system eventually stabilizes and returns to satisfy φ_s after a finite number of violations

This work was supported by the Graz University of Technology's LEAD project "Dependable Internet of Things in Adverse Environments". It was also partially supported by the Coleman-Cohen Foundation, and by the Royal Society grant IES\R2\170021.

of φ_e's safety assumptions. A similar definition also appeared in [7]: 'bounded disturbances lead to bounded deviations from nominal behavior,' and 'the effect of a sporadic disturbance disappears in finitely many steps'.

(R6) According to [4], a robust system satisfies its liveness guarantees, even if φ_e is violated infinitely often.

(R7) k-robustness (φ_e): A robust system satisfies φ_s even if the environment violates φ_e up to k consecutive times.

The above definitions of robustness can be further generalized by adding a dimension of a *recovery specification*, which is an LTL formula that defines how the system recovers from failures. For example, **(R5)** with the dimension of recovery can specify that the system should recover within 5 steps after each failure of the environment to satisfy φ_e. We denote the extensions of the above robustness definitions with recovery by replacing **R** with **C**, e.g., **(R5)** becomes **(C5)**.

For completeness, we should mention that there is an additional definition of robustness in [6], based on a quantitative measure, which our framework is unable to accommodate.

We note that the informal definitions in **(R5)** and **(R7)** generally cannot be applied to environment assumptions in LTL, because such formulas are interpreted over infinite traces with respect to the initial state. For example, if $\varphi_e = \mathbf{G}\,\mathbf{F}\,r$, then we can say whether a given path satisfies it or not, but there is no meaning to saying that it violates φ_e k times or even a finite number of times. The same problem exists if the guarantees are in general LTL: it is meaningless to say that they are violated for a finite number of times.

We therefore interpret these informal definitions as relevant only for assumptions and guarantees of the form $\mathbf{G}\,\psi$, where ψ is a propositional formula, and the failure count refers to the number of locations along the path that satisfy $\neg\psi$.

In the next two sections we will present our suggested method for synthesizing robust systems against hidden inputs and assumptions violations, respectively.

III. Synthesis with a robustness specification: Hidden Inputs

The key ingredient of our method is what we call a *synchronization bit* — a single bit $\boldsymbol{sb}$ that indicates whether the inputs are hidden in the current step. $\boldsymbol{sb}$ is an additional input to the synthesized system M. We will demonstrate how to synthesize a robust system while using this bit, by focusing on the robustness definitions that focus on hidden inputs: **(R1)**, **(R2)** and **(R3)**.

As a first step, we define the temporal behavior of $\boldsymbol{sb}$ so it captures our definition of robustness. Consider, for example, **(R1)**. Recall that it defines a system to be robust if it satisfies φ_s even if the inputs are hidden for a finite number of steps. We can translate this definition into an LTL *robustness specification* φ_{rob} over $\boldsymbol{sb}$:

$$\varphi_{rob}(\boldsymbol{sb}) = \mathbf{F}\,\mathbf{G}\,\boldsymbol{sb}\ . \tag{2}$$

Similarly, **(R2)** is captured by the robustness specification

$$\mathbf{G}(\overline{\boldsymbol{sb}} \rightarrow (\mathbf{X}\,\boldsymbol{sb} \vee \ldots \vee \mathbf{X}^k\,\boldsymbol{sb}))\ , \tag{3}$$

i.e., the inputs are hidden up to k steps consecutively. **(R3)** is the most general definition and therefore as a special case can be Eq. (2) or Eq. (3), but also other specifications such as

$$\varphi_{rob}(\boldsymbol{sb}) = \mathbf{G}\,\mathbf{F}(\overline{\boldsymbol{sb}}) \tag{4}$$

(there will be misreadings of the inputs infinitely often) or

$$\varphi_{rob}(\boldsymbol{sb}) = (in_1 = in_2 \rightarrow \boldsymbol{sb}) \tag{5}$$

(we trust the inputs only if they are equal). A plausible scenario is when the relation between the inputs is an indication of whether they can be trusted. For example, when using *error-correcting code*, the relation between bits in a packet and a parity bit at its end reveals whether to trust the packet.

As a second step, we change the specification so it operates on free inputs when $\boldsymbol{sb}$ is low. Let I denote the set of φ's inputs that are considered as hidden, and I' an additional copy of those inputs. We attempt to synthesize:

$$[\varphi_{rob}(\boldsymbol{sb}) \wedge \mathbf{G}(\boldsymbol{sb} \rightarrow (I = I')) \wedge \varphi_e(I, O)] \rightarrow \varphi_s(I', O)\ . \tag{6}$$

A system realizing (6) operates over the extended set of inputs $I \cup I' \cup \{\boldsymbol{sb}\}$, and not over the original set of inputs I. Note that $\varphi_s(I', O)$ is over I' (and not I), and that the assumption $\varphi_e(I, O)$ is over the original inputs I. Only when $\boldsymbol{sb}$ is high, I' is bound to be equal to I and hence adheres to the assumptions. The signals I and sb are not visible to the system. We thus have a problem of synthesis with *incomplete information*, which can be handled in double exponential time (see [10]), like standard synthesis.

The following example, however, illustrates a case where synthesizing M according to (6) does not result in a system that realizes the original specification.

Example 1. Let $\varphi_e = \textbf{true}$ and $\varphi_s = \mathbf{G}(r \leftrightarrow g)$, where $I = \{r\}$. Let $\varphi_{rob} = \overline{\boldsymbol{sb}} \wedge \mathbf{X}\,\mathbf{G}\,\boldsymbol{sb}$ ($\boldsymbol{sb}$ is down in the first state and is up in all other states). Any system M realizing (6) contains, in particular, paths starting with the state $(\bar{r}, r', g)$, which clearly do not satisfy $\mathbf{G}(r \leftrightarrow g)$. The system in Fig. 1 demonstrates it. The top label from state 0 to state 1 is such a transition (r is not part of the label because it can have any value, and in particular $\bar{r}$). □

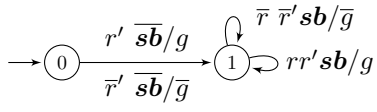


Fig. 1: M for Example 1

We would like the result of our synthesis to be robust according to the following definition:

Definition 1. A system M is *robust w.r.t. a set of hidden inputs I and a robustness specification φ_{rob}* if it realizes both (6) and the original specification $\varphi_e(I, O) \rightarrow \varphi_s(I, O)$.

The next question we address is what are sufficient conditions on φ_s and φ_e for M to be robust according to Def. 1.

We say that an LTL formula $\varphi(X)$ over a set of variables X is *monotone increasing* in $x \in X$ iff for every two paths π_1

and π_2 that agree with each other on the values of $X \setminus \{x\}$, and their labeling with x is such that

$$\forall j \geq 0.(\pi_1[j] \models x) \rightarrow (\pi_2[j] \models x), \tag{7}$$

we have

$$(\pi_1 \models \varphi) \rightarrow (\pi_2 \models \varphi). \tag{8}$$

Similarly, $\varphi(X)$ is *monotone decreasing in* x if (7) implies $(\pi_2 \models \varphi) \rightarrow (\pi_1 \models \varphi)$.

Let us now consider a path π in M. It is labeled with $I \cup I' \cup \{\boldsymbol{sb}\} \cup O$. For simplicity assume that I consists of one input i, i.e., $I = \{i\}$, and hence $I' = \{i'\}$. We show a sufficient condition for M to satisfy the conditions of Def. 1.

Lemma 1. *If $\varphi_s(I, O)$ is monotone increasing in i and*

$$\varphi_{rob}(\boldsymbol{sb}) \wedge \varphi_e(i, O) \wedge \mathbf{G}(\boldsymbol{sb} \rightarrow (i' = i)) \models \mathbf{G}(i' \rightarrow i), \tag{9}$$

then any system M that realizes (6) *is robust w.r.t. hidden inputs (see Def. 1).*

Proof. We define ρ as the projection of π to $\{i'\} \cup O$ and rename i' with i. Because of the renaming we have

$$\forall j \geq 0.(\pi[j] \models i') \longleftrightarrow (\rho[j] \models i). \tag{10}$$

(9) implies that for each path $\pi \in M$ that satisfies $\varphi_e(i, O)$,

$$\forall j \geq 0.(\pi[j] \models i') \rightarrow (\pi[j] \models i). \tag{11}$$

Now, (10) and (11) imply

$$\forall j \geq 0.(\rho[j] \models i) \rightarrow (\pi[j] \models i), \tag{12}$$

and therefore ρ and π satisfy (7). Recall that $\pi \in M$ and $\pi \models \varphi_e(i, O)$. Hence $\pi \models \varphi_s(i', O)$, and thus $\rho \models \varphi_s(i, O)$. Due to the assumed monotonicity of φ_s and (8)

$$(\rho \models \varphi_s(i, O)) \rightarrow (\pi \models \varphi_s(i, O)). \tag{13}$$

Recall that π is any path of M that satisfies $\varphi_e(i, O)$, thus (13) shows that all paths of M that satisfy $\varphi_e(i, O)$ also satisfy $\varphi_s(i, O)$, and therefore M is robust according to Def. 1. □

Corollary 1. *The following is an immediate corollary from Lemma 1. If $\varphi_s(I, O)$ is monotone* decreasing *in i and*

$$\varphi_{rob}(\boldsymbol{sb}) \wedge \varphi_e(i, O) \wedge \mathbf{G}(\boldsymbol{sb} \rightarrow (i' = i)) \models \mathbf{G}(i \rightarrow i'), \tag{14}$$

then any system M that realizes (6) *is robust w.r.t. hidden inputs (see Def. 1).*

Example 2. Let $\varphi_e = \overline{r} \wedge \mathbf{X}\,\overline{r}$, $\varphi_s = \mathbf{G}(r \rightarrow g)$, where $I = \{r\}$, and $\varphi_{rob} = \overline{\boldsymbol{sb}} \wedge \mathbf{X}\,\mathbf{G}\,\boldsymbol{sb}$. It is easy to see that φ_s is decreasing in r. The system presented in Fig. 2 realizes both (6) and $\varphi_e \rightarrow \varphi_s$, in line with Corollary 1. □

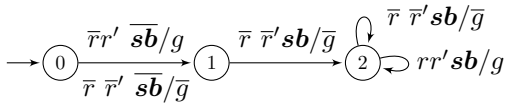


Fig. 2: M for Example 2. All transitions not shown lead to a 'sink' state emitting $\bar{g}$, omitted here for clarity.

A natural question is how to test whether (9) or (14)) hold, given φ_{rob} and φ_e. This problem can be reduced to the question of whether (9) can be falsified, or in other words to the question of whether the following formula is satisfiable:

$$\varphi_{rob}(\boldsymbol{sb}) \wedge \varphi_e(i, O) \wedge \mathbf{G}(\boldsymbol{sb} \rightarrow (i' = i)) \wedge \mathbf{F}(i' \wedge \neg i)\ . \tag{15}$$

In our implementation of this method in PARTY, the user specifies φ_{rob}, φ_e and φ_s, and the rest is done automatically. But it is clear that any reactive synthesizer can be used as is by rewriting the specification according to (6).

IV. Synthesis with a robustness specification: Environment Assumptions

Let us now focus on robustness definitions that refer to scenarios in which the environment assumptions cannot always be trusted. Generally it is impossible to synthesize systems that are robust according to definitions such as (**R7**), where the number of violations is bound to be finite or lower than some number k, assuming φ_e and φ_s are general LTL formulas. The reason is that such formulas are either satisfied by a path or not, and hence there is no meaning to counting the number of violations. A similar conclusion was reached by Ehlers in [5].

We can, however, restrict ourselves to invariant properties, i.e., formulas of the form $\mathbf{G}\,\psi$ where ψ is propositional.[1] As in [5], we then interpret 'failure' as a temporary failure of the invariant. Hence the failure-count refers to the number of time frames in which φ is false. Our approach is different than [5] because we have the freedom to specify the robustness specification φ_{rob}, whereas [5] suggested an algorithm that is tailored for systems that are eventually reliable, i.e., $\varphi_{rob} = \mathbf{F}\,\mathbf{G}\,\boldsymbol{sb}$. In addition, our method works with any existing synthesizer, whereas [5] requires a tailored algorithm (based on a generalized Rabin(1) automaton).

Let $\varphi_e \equiv \mathbf{G}\,\psi_e$ and $\varphi_s \equiv \mathbf{G}\,\psi_s$, where ψ_e, ψ_s are propositional. The following are several examples of formulas involving a synchronization bit; the synthesis of each one of them, when possible, results in a system that satisfies the original specification $\varphi_e \rightarrow \varphi_s$, each time with a different level of robustness.

$$(\varphi_{rob}(\boldsymbol{sb}) \wedge \mathbf{G}(\boldsymbol{sb} \rightarrow \psi_e)) \rightarrow (\mathbf{G}\,\psi_s). \tag{16}$$

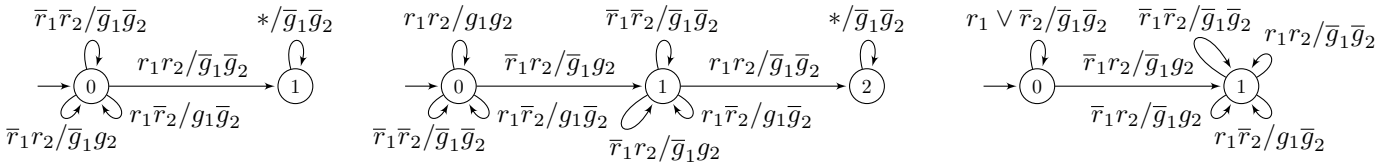
A system satisfying (16) is robust according to (**R4**), since it satisfies ψ_s even in the presence of failures of the environment.

$$\varphi_{rob}(\boldsymbol{sb}) \rightarrow \mathbf{G}((\boldsymbol{sb} \wedge \psi_e) \rightarrow \psi_s). \tag{17}$$

A system satisfying (17) is robust according to (**R5**), since it satisfies ψ_s when $\boldsymbol{sb}$ is high.

Example 3. Consider Spec. 1. Figure 3a shows a non-robust system realizing the aforementioned specification. As shown,

[1]The subformula ψ can also be temporal, but then it is less clear what is the meaning of failure. For example suppose that $\varphi_e = \mathbf{G}\,\mathbf{F}\,p$ and that at a certain step φ_e fails. This means that φ_e fails in all future steps as well, since from that step onward, p cannot happen infinitely often. For this reason here we restrict ourselves to ψ which is propositional.



(a) M_1 without robustness (b) M_2 with robustness w.r.t. (16) (c) M_3 with robustness w.r.t. (17)

Fig. 3: Systems for Spec. 1 with different levels of robustness. The value of $\boldsymbol{sb}$ is omitted from the drawings. In (a), where there is no robustness specification, the system transits to a trap state once the assumption is violated. In that state it no longer satisfies φ_s. System (b) was synthesized according to (16) and φ_{rob} that was described in (19). Note that as long as $\boldsymbol{sb}$ is down (at state 0, which is before the password was given) violating the assumption does not lead us to a sink state, and the guarantee still holds. This is in contrast to (c), which is synthesized with (17), where a violation of the assumption invariant can lead to a violation of the guarantee's invariant (indeed the input r_1r_2 at state 0 emits $\overline{g}_1\overline{g}_2$ which contradicts φ_s).

Assume	Guarantee
$\mathbf{G}(\overline{r}_1 \vee \overline{r}_2)$	$\mathbf{G}(r_1 \rightarrow g_1) \wedge$ $\mathbf{G}(r_2 \rightarrow g_2)$

Spec. 1: A specification for an immediate arbiter of two clients that assumes requests are mutually exclusive.

once the assumption fails, the system transits to a 'sink' state in which it does not satisfy φ_s.

We now demonstrate the effect of adding a robustness specification according to (16) and (17), which happens to depend on the relation between the inputs. Suppose that Spec. 1 works in an adversarial environment, and we want it to start trusting the inputs only after it receives some password confirming the identity of the environment. This pattern can be specified with

$$\varphi_{rob} = (\neg pwd \wedge \overline{\boldsymbol{sb}})\, \mathbf{W}(pwd \wedge \mathbf{G}(\boldsymbol{sb}))\ , \tag{18}$$

where pwd is the password, and $\mathbf{W}$ is the weak-until operator. For this example assume that $pwd = \overline{r}_1 \wedge r_2$, and hence the robustness specification becomes

$$\varphi_{rob} = (\neg(\overline{r}_1 \wedge r_2) \wedge \overline{\boldsymbol{sb}})\, \mathbf{W}((\overline{r}_1 \wedge r_2) \wedge \mathbf{G}(\boldsymbol{sb}))\ . \tag{19}$$

The systems depicted in Figs. 3b and 3c are the results of (16) and (17), respectively. See the caption for more information. □

Eq. (17) can be generalized to the case where we have a recovery specification φ_{rec} in addition to φ_{rob}.

As an example, suppose that we allow the system to fault for k time steps after a failure of the environment to satisfy φ_e. In such a case we will specify

$$\varphi_{rec} = \mathbf{G}((\boldsymbol{sb} \wedge \mathbf{X}\,\boldsymbol{sb} \wedge \ldots \wedge \mathbf{X}^k\,\boldsymbol{sb}) \rightarrow \mathbf{X}^k\,\boldsymbol{sb'})\ . \tag{20}$$

It is left to specify the behavior of $\boldsymbol{sb'}$ in the first k steps, for example we may require that in step i for $i \leq k$, $\boldsymbol{sb'}$ should be high if in steps $0 \leq j \leq i$ $\boldsymbol{sb}$ was high (not shown).

Given φ_{rec}, we synthesize the following:

$$(\varphi_{rob}(\boldsymbol{sb}) \wedge \varphi_{rec}(\boldsymbol{sb}, \boldsymbol{sb'})) \rightarrow \mathbf{G}((\boldsymbol{sb} \wedge \psi_e) \rightarrow (\boldsymbol{sb'} \rightarrow \psi_s)). \tag{21}$$

A system satisfying (21) is robust according to **(C5)**, since (21) generalizes (17) by requiring the system to satisfy the guarantee only in the steps described by the recovery specification φ_{rec}. The following example demonstrates the result of adding the recovery specification (20) with $k = 1$.

Example 4. Fig. 4 shows a system satisfying Spec. 1 with the robustness specification (19) and the recovery specification

$$\varphi_{rec} \equiv \mathbf{G}((\boldsymbol{sb} \wedge \mathbf{X}\,\boldsymbol{sb}) \rightarrow \mathbf{X}\,\boldsymbol{sb'})\ . \tag{22}$$

□

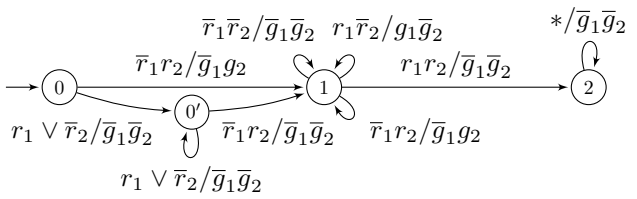


Fig. 4: A system satisfying Spec. 1, with the robustness specification (19) and the recovery specification (22). Note that after a violation of the assumption with r_1r_2 at state 0 the system transits to state $0'$ with an arbitrary output and then has an additional transition with an arbitrary output. In the diagram the output is forced to $\overline{g}_1\overline{g}_2$ because we chose to synthesize a least-vacuous system in PARTY. Without this option any output on these transitions is legal.

V. CONCLUSION

The incompleteness of most specifications gives a certain freedom to the synthesis algorithm, and it is left for the designer of the synthesis algorithm to set criteria for what system is preferable, from the large set of options. Several previous publications suggested to use this freedom in order to synthesize a system which is robust. Each of the publications had its own definition of robustness, and each suggested a synthesis method to achieve a robust system. In this article we showed a general framework that enables to specify with an LTL formula φ_{rob} the expected robustness, and from there on it reduces the robust synthesis problem to that of normal synthesis. Furthermore, we enable to specify the recovery specification φ_{rec} with an LTL formula. Our framework is flexible and easy to use with existing reactive synthesizers, and, as we showed, it covers most of the previously published definitions of robustness, as well as several new ones that we introduced here. Our tool PARTY, which implements our method, is available for download from https://www.iaik.tugraz.at/content/research/opensource/eris/.

References

[1] U. B. S. Almagor and O. Kupferman, "Formalizing and reasoning about quality," in *ICALP*, 2013, p. 15–27.

[2] R. Ehlers, R. Könighofer, and G. Hofferek, "Symbolically synthesizing small circuits," in *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*, G. Cabodi and S. Singh, Eds. IEEE, 2012, pp. 91–100.

[3] G. Jing, R. Ehlers, and H. Kress-Gazit, "Shortcut through an evil door: Optimality of correct-by-construction controllers in adversarial environments," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*. IEEE, 2013, pp. 4796–4802. [Online]. Available: https://doi.org/10.1109/IROS.2013.6697048

[4] R. Bloem, H. Gamauf, G. Hofferek, B. Könighofer, and R. Könighofer, "Synthesizing robust systems with RATSY," in *Proceedings First Workshop on Synthesis, SYNT 2012, Berkeley, California, USA, 7th and 8th July 2012.*, ser. EPTCS, D. A. Peled and S. Schewe, Eds., vol. 84, 2012, pp. 47–53. [Online]. Available: https://doi.org/10.4204/EPTCS.84.4

[5] R. Ehlers, "Generalized Rabin(1) synthesis with applications to robust system synthesis," in *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, ser. Lecture Notes in Computer Science, M. G. Bobaru, K. Havelund, G. J. Holzmann, and R. Joshi, Eds., vol. 6617. Springer, 2011, pp. 101–115.

[6] R. Bloem, K. Chatterjee, K. Greimel, T. A. Henzinger, G. Hofferek, B. Jobstmann, B. Könighofer, and R. Könighofer, "Synthesizing robust systems," *Acta Informatica*, vol. 51, no. 3-4, pp. 193–220, 2014.

[7] S. Y. C. Y. S. P. Tabuada, A. Balkan and R. Majumdar, "Input-output robustness for discrete systems," in *EMSOFT*, 2012, p. 217–226.

[8] U. Topcu, N. Ozay, J. Liu, and R. M. Murray, "On synthesizing robust discrete controllers under modeling uncertainty," in *Hybrid Systems: Computation and Control (part of CPS Week 2012), HSCC'12, Beijing, China, April 17-19, 2012*, T. Dang and I. M. Mitchell, Eds. ACM, 2012, pp. 85–94.

[9] "ISO/IEC/IEEE International Standard - Systems and Software Engineering – Vocabulary," *ISO/IEC/IEEE 24765:2010(E)*, pp. 1–418, Dec 2010.

[10] O. Kupferman and M. Vardi, "Synthesis with incomplete informatio," in *2nd International Conference on Temporal Logic*, Manchester, July 1997, pp. 91–106.

[11] A. Khalimov, S. Jacobs, and R. Bloem, "PARTY parameterized synthesis of token rings," in *CAV*, ser. LNCS, N. Sharygina and H. Veith, Eds., vol. 8044. Springer, 2013, pp. 928–933.