Resilience to Intermittent Assumption Violations in Reactive Synthesis

Rüdiger Ehlers
University of Bremen
Bremen, Germany

University of Kassel
Kassel, Germany

Ufuk Topcu
University of Pennsylvania
Philadelphia, Pennsylvania, United States

ABSTRACT

We consider the synthesis of reactive systems that are robust against intermittent violations of their environment assumptions. Such assumptions are needed to allow many systems that work in a larger context to fulfill their tasks. Yet, due to glitches in hardware or exceptional operating conditions, these assumptions do not always hold in the field. Manually constructed systems often exhibit *error-resilience* and can continue to work correctly in such cases. With the development cycles of reactive systems becoming shorter, and thus reactive synthesis becoming an increasingly suitable alternative to the manual design of such systems, automatically synthesized systems are also expected to feature such resilience.

The framework for achieving this goal that we present in this paper builds on generalized reactivity(1) synthesis, a synthesis approach that is well-known to be scalable enough for many practical applications. We show how, starting from a specification that is supported by this synthesis approach, we can modify it in order to use a standard generalized reactivity(1) synthesis procedure to find error-resilient systems. As an added benefit, this approach allows exploring the possible trade-offs in error resilience that a system designer has to make, and to give the designer a list of all Pareto-optimal implementations.

1. INTRODUCTION

Automatically synthesizing reactive systems from their specifications is an attractive alternative to constructing these by hand. Even when a complete specification is not available, formal synthesis is a useful approach to analyze the specifications for the parts of the system to be constructed and allows to explore the design alternatives in a structured way.

To fully benefit from synthesis technology, measures have to be taken to ensure that the implementations computed in the process are of good quality [1, 7, 11]. Example quality criteria include energy consumption, size of the implementation, and the resilience of the implementation against changes in the conditions under which the system operates. Intuitively, the latter criterion means that a system should work as well as possible in scenarios in which the assumptions about its environment are violated. In other words, the system shall degrade gracefully, and all safety-relevant properties of the system should be fulfilled "whenever" possible.

While it would obviously be best if a synthesized system does not rely on any environment assumptions being satisfied, this is typically not possible. For example, if we require a robot to go from one point in a workspace to another point and there is an obstacle in between, then the assumption that the position of the robot is updated according to its actions (move left, move right, etc.) needs to be made. Manually constructed reactive system controllers typically only rely on these environment assumptions being satisfied in situations in which they are crucially needed. A reactive synthesis procedure on the other hand will typically compute a controller that lets it come very close to the obstacle, and thus the resulting controller cannot even tolerate a single "glitch" in the environment assumptions. The reason for this is by default, reactive synthesis procedures do not optimize towards controller behavior that allows for error-resilience (such as staying away from an obstacle as far as possible). Even worse, once an assumption is violated, the system is free to behave in an arbitrary manner and in particular, possibly fail to fulfill its requirements. The violation of assumptions in the field does also not necessarily mean that they have been modeled incorrectly, as typically not all eventualities can be modeled correctly and precisely, like components of the robot breaking at runtime or dirt on sensors leading to imperfect measurements.

In this paper, we solve the problem of synthesizing error-resilient systems from specifications in temporal logic. We concentrate on the generalized reactivity(1) fragment of linear-time temporal logic (LTL), for which an efficient and symbolic synthesis algorithm is known [4]. We show how to add the requirement of being k-resilient [10] to such a specification. That is the system has to tolerate arbitrarily many violations of safety assumptions ("*glitches*"), as long as in between every k such glitches, there is a long enough period in which no glitch occurs so that the system can recover from the earlier k glitches. By not exceeding the class of generalized reactivity(1) specification in this process, we ensure that also the synthesis problem for the resulting specification can be solved efficiently.

HSCC'14, April 15–17, 2014, Berlin, Germany.

ACM 978-1-4503-2732-9/14/04 ...$15.00.
http://dx.doi.org/10.1145/2562059.2562128.

Automatically synthesizing error-resilient systems enables to effectively perform *design space exploration*: we compute (1) which assumptions need to be seen as strict, i.e., need to be satisfied all of the time, (2) for which assumptions arbitrarily many glitches can be tolerated, and (3) for which assumption some glitches can be tolerated, and whose violations should count towards the value of k. We present an exploration algorithm that searches for all Pareto-optimal assignments of the assumptions to these categories that represent implementable error-resilience guarantees. Searching for these optimal solutions gives the system designer the insight of what the specifications imply with respect to the system's resilience and thus to select the most reasonable solution for the practical application in mind, without the need to formalize the preferences in advance. Additionally, it allows the system designer to state the assumptions in a very conservative manner whenever a precise model of the assumptions cannot be given. In the robot example, we might for instance know that dislocations of the robot do not happen "very often," but a more precise characterization of the environment cannot be given. By assuming that these never happen, and applying error-resilient synthesis, we can explore the possible error-resilience levels with our synthesis approach and by picking the most suitable one save the effort to analyze the possible environment behavior more closely in order to obtain a most suitable controller.

The rest of the paper is structured as follows: In the next section, we recall some preliminaries. Then, we formally describe the error-resilient system synthesis problem in Section 3, and explain how a generalized reactivity(1) specification can be modified in order to add error-resilience as a requirement in Section 4. We state how to search for all Pareto-optimal solutions in Section 5 and show the usefulness of our new techniques experimentally in Section 6. Afterwards, we discuss related work in Section 7 and conclude with a summary in Section 8.

2. PRELIMINARIES

A *controller* is a reactive system with an *interface* $\mathcal{I} = (\mathsf{AP}_I, \mathsf{AP}_O)$, where AP_I is the set of *input signals* and AP_O is the set of *output signals*. Given an interface $\mathcal{I}$, we describe a controller as a finite-state machine $\mathcal{M} = (S, \mathsf{AP}_I, \mathsf{AP}_O, s_0, \delta)$ for which S is a finite set of states, $s_0 \in S$ is the initial state, and $\delta : S \times 2^{\mathsf{AP}_I} \to S \times 2^{\mathsf{AP}_O}$ is the transition function. We say that some word $w = (w_0^I, w_0^O)(w_1^I, w_1^O)(w_2^I, w_2^O) \ldots \in (2^{\mathsf{AP}_I} \times 2^{\mathsf{AP}_O})^\omega$ is a *trace* of $\mathcal{M}$ if there exists a *run* $\pi = \pi_0\pi_1 \ldots \in S^\omega$ such that $\pi_0 = s_0$ and for every $i \in \mathbb{N}$, we have $(\pi_{i+1}, w_i^O) = \delta(\pi_i, w_i^I)$. We call a sequence of states $s_1 s_2 \ldots s_n \in S$ a *loop* in $\mathcal{M}$ if $s_1 = s_n$ and for all $i \in \{1, \ldots, n-1\}$, there exist $x \subseteq \mathsf{AP}_I$ and $y \subseteq \mathsf{AP}_O$ such that $(s_{i+1}, y) = \delta(s_i, x)$.

In *reactive synthesis*, we compute a finite-state machine for which all of its traces satisfy some specification, which we describe in *temporal logic*. We consider a subset of *linear-time temporal logic (LTL)* [13] in this paper. Formulas in LTL are evaluated at positions i in a word $w = w_0 w_1 \ldots \in (2^{\mathsf{AP}})^\omega$ over a set of atomic propositions AP. In addition to the standard Boolean operators, we have the temporal operators G, F, and X that connect the truth values at some position in a word to those at future positions. Formally, we have (1) $w, i \models \mathsf{X}\phi$ iff $w, (i+1) \models \phi$, (2) $w, i \models \mathsf{G}\phi$ iff $\forall j \geq i : w, j \models \phi$, and (3) $w, i \models \mathsf{F}\phi$ iff $\exists j \geq i : w, j \models \phi$ (for every trace w, index i and sub-formula ϕ). The *until* operator of LTL is not used in this paper and not defined here. We say that a finite-state machine $\mathcal{M} = (S, \mathsf{AP}_I, \mathsf{AP}_O, s_0, \delta)$ satisfies some LTL specification ψ if and only if all of its traces (over $\mathsf{AP} = \mathsf{AP}_I \cup \mathsf{AP}_O$) satisfy the LTL formula at position $i = 0$. Given an interface $\mathcal{I} = (\mathsf{AP}_I, \mathsf{AP}_O)$ and an LTL specification ψ over the set of atomic propositions $\mathsf{AP}_I \cup \mathsf{AP}_O$, the *synthesis problem* is to decide whether there exists a finite-state machine $\mathcal{M} = (S, \mathsf{AP}_I, \mathsf{AP}_O, s_0, \delta)$ that satisfies ψ, and to compute such a finite-state machine in case of a positive answer.

A particularly interesting sub-class of LTL for the purpose of synthesis is the class of *generalized reactivity(1)* specifications [4], which is abbreviated as *GR(1)* in the following. Such specifications are of the form

$$\psi = (\psi_i^a \wedge \psi_s^a \wedge \psi_l^a) \rightarrow (\psi_i^g \wedge \psi_s^g \wedge \psi_l^g),$$

where the parts left of the implication operator are the *assumptions* and the parts right of the implication operator are the *guarantees*. In both of these property groups, we have *initialization properties* (ψ_i^a and ψ_i^g), which are free of temporal operators, *safety properties* (ψ_s^a and ψ_s^g), which are of the form $\mathsf{G}\phi$ for some LTL sub-formula ϕ in which the only temporal operator allowed is X, and *liveness properties* (ψ_l^a and ψ_l^g), which are of the form $\mathsf{GF}\phi$ for some LTL sub-formula ϕ in which the only temporal operator allowed is X. In all of these properties, the X operator is not allowed to be nested. In ψ_s^a and ψ_l^a, variables from AP_O may not occur within the scope of an X operator. Synthesizing from GR(1) specifications can be performed in time exponential in $|\mathsf{AP}_I| + |\mathsf{AP}_O|$, and tools that circumvent this worst-case computation time for many practical cases have been developed [12, 6]. It must be noted however that these typically implement a *strict semantics* of GR(1), where in order for a system to be considered to satisfy ψ, we have that ψ_s^g may only be violated after ψ_s^a has already been violated. This choice has no conceptual reasons – a specification for the classical Boolean implication semantics between the assumptions and guarantees can be encoded into one for the strict semantics with a minor blow-up of adding only one atomic proposition. For more details, the reader is referred to Bloem et al. [4]. In contrast to older GR(1) synthesis literature, we also allow the next-time operator in liveness assumptions and guarantees, which has been shown to only require a minor change to the standard GR(1) synthesis procedure [14].

3. DEFINING ERROR-RESILIENCE

In this section, we introduce a series of notions of resilience and state the main problems solved in the subsequent sections. Let AP be the set of signals of a reactive system and $\psi = \psi^a \rightarrow \psi^g = (\psi_i^a \wedge \psi_s^a \wedge \psi_l^a) \rightarrow (\psi_i^a \wedge \psi_s^a \wedge \psi_l^a)$ be its specification. Recall that ψ_s^a is a conjunction of basic safety properties, i.e., we have $\psi_s^a = \psi_{s,1}^a \wedge \ldots \wedge \psi_{s,n}^a$ with $\psi_{s,j}^a = \mathsf{G}\phi_{s,j}^a$ for all $j \in \{1, \ldots, n\}$.

3.1 Uniform Error-Resilience

Let $w = w_0 w_1 w_2 \ldots \in (2^{\mathsf{AP}})^\omega$ be a trace of a reactive system. We say that we have an *intermittent assumption violation at some position* $i \in \mathbb{N}$ for property $\psi_{s,j}^a = \mathsf{G}\phi_{s,j}^a$ in the trace if $w_{i-1} w_i w' \not\models \phi_{s_j}^a$ for every $w' \in (2^{\mathsf{AP}})^\omega$. For simplicity, let us also call intermittent assumption violations *glitches*. We say that $l \in \mathbb{N}$ glitches occur at some position

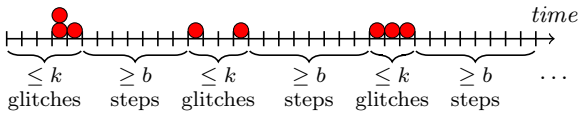

Figure 1: Graphical description of a (k, b)-sane input stream for some system and $(k, b) = (3, 6)$; glitches along the system's execution are marked by circles.

$i \in \mathbb{N}$ in w if the maximal subset $P \subseteq \{\psi^a_{s,1}, \ldots, \psi^a_{s,n}\}$ such that for every $\psi' \in P$, we have a glitch for ψ' at position i, has cardinality l.

Definition 1. *Let $(\psi^a_i \wedge \psi^a_s \wedge \psi^a_l) \rightarrow \psi^g$ be a specification for some interface $\mathcal{I} = (\mathsf{AP}_I, \mathsf{AP}_O)$ and $k, b \in \mathbb{N}$. We say that some trace w is* (k, b)-sane *if in w,*

1. *there are infinitely many sequences of at least b consecutive steps in which no glitch occurs,*
2. *in between every of these glitch-free sequences, there are at most k glitches, and*
3. *there are only finitely many glitches in total.*

We say that a reactive system $\mathcal{M}$ with the interface $\mathcal{I}$ is (k, b)-resilient *if every of its traces satisfies ψ^g if it is (k, b)-sane and satisfies ψ^a_i and ψ^a_l.*

Figure 1 explains the concept graphically. Note that our definition of glitches is concerned with both the input and output part of a trace, which may be a bit unintuitive at first. It allows us to define the allowed *next* input signal valuations in the safety assumptions based on the system's previous output, however.

Intuitively, if a system is (k, b)-resilient, then it can from time to time tolerate a sequence of up to k intermittent assumption violations (not necessarily consecutive), provided that there are at least b steps to recover from the glitches afterwards. The restriction to finitely many glitches ensures that in cases in which the environment of a reactive system uses the glitches to prevent the system from fulfilling its liveness objectives, the system does not need to satisfy its guarantees. Since the system may have to wait for progress with respect to the environment satisfying its liveness assumptions before it can make progress towards satisfying the liveness guarantees, a recovery period of b steps may be insufficient for any value of b in the case of infinitely many glitches. The restriction to finitely many glitches does not affect the idea of error resilience, however, as a system can never know when a glitch has been the last one. Thus, it always has to return to normal operation mode after some finite number of steps and has to work towards satisfying its liveness guarantees even in the case of infinitely many glitches.

Note that we excluded the initialization assumptions and the liveness assumptions from being considered in the definition of error-resilience. The reason is simply that we want to make the system under design robust against *intermittent* violations of the assumptions. Liveness and initialization assumptions cannot be violated temporarily. If we are able to synthesize a reactive system even for the case that one such assumption is violated, then it is not needed. It can be assumed that in a formal development process of a reactive system, in which a specification is built step-by-step, such an assumption would never be added, so not considering this case does not restrict the practical impact of our error-resilient synthesis techniques to follow.

We are now ready to define the error-resilient synthesis problems.

Definition 2. *Given a specification $\psi = \psi^a \rightarrow \psi^g$, an interface $\mathcal{I} = (\mathsf{AP}_I, \mathsf{AP}_O)$, and values $k, b \in \mathbb{N}$, we define the reactive (k, b)-resilient synthesis problem for ψ and $\mathcal{I}$ to check if there exists a (k, b)-resilient system for $\mathcal{I}$ that satisfies ψ, and to compute such a system whenever it exists. In the reactive k-resilient synthesis problem [10] for $\mathcal{I}$ and ψ, we want to find a system for $\mathcal{I}$ whose traces satisfy ψ and that is (k, b)-resilient for some $b \in \mathbb{N}$, if it exists.*

3.2 Mixed Error-Resilience

The k-resilience definition introduced above treats all (safety) assumptions alike. In practical specifications, they however often differ in how likely they are to be violated at runtime. For example, for a robot, the assumption that the sensed position does not jump from one end of the workspace to another one is less likely to be violated compared to the assumption that the robot position is always updated precisely according to the actuators involved, as people can bump into the robot and thus cause small dislocations. As we cannot expect the robot to work correctly if people bump into it too often, this assumption is nevertheless justified and typically also needed. To account for these different likelihoods of intermittent assumption violations, we now refine the idea of k-resilience accordingly.

Definition 3. *Let $\psi^a_s = \psi^a_{s,1} \wedge \ldots \wedge \psi^a_{s,n}$ be the set of safety assumptions of a given specification. We define a* resilience signature *for ψ^a_s to be a function $s : \{1, \ldots, n\} \rightarrow \{any, some, none\}$.*

The idea of a resilience signature is that it augments a reactive system specification with information about the relative significance of the violations in its safety assumptions with respect to its resilience. More specifically, it classifies the assumptions as those for which (1) any number of glitches shall be tolerated, (2) a particular pre-specified number of glitches shall be tolerated, or (3) no glitches are guaranteed to be tolerated. Only case (2) shall count toward the value of k that we define resilience over.

Definition 4. *Let $(\psi^a_i \wedge \psi^a_s \wedge \psi^a_l) \rightarrow \psi^g$ be a specification for some interface $\mathcal{I} = (\mathsf{AP}_I, \mathsf{AP}_O)$, s be a resilience signature for ψ^a_s, and $k, b \in \mathbb{N}$. We say that some trace w is* (s, k, b)-sane *if in w,*

1. *there are infinitely many sequences of at least b consecutive steps in which no glitch occurs,*
2. *no glitches occur at all for all safety assumptions $\psi^a_{s,j}$ with $s(\psi^a_{s,j}) = none$,*
3. *in between every of these glitch-free sequences, there are at most k glitches of safety assumptions $\psi^a_{s,j}$ with $s(\psi^a_{s,j}) = some$, and*
4. *there are only finitely many glitches in total.*

We say that a reactive system $\mathcal{M}$ is (s, k, b)-resilient *if every trace of the system satisfies ψ^g if the trace is (s, k, b)-sane and satisfies ψ^a_i and ψ^a_l.*

The resilient synthesis problem can now be modified accordingly as follows.

DEFINITION 5. *Given a specification $\psi = \psi^a \rightarrow \psi^g$, an interface $\mathcal{I} = (\mathsf{AP}_I, \mathsf{AP}_O)$, values $b, k \in \mathbb{N}$, and some resilience signature s, we define the reactive (s,k,b)-resilient synthesis problem for ψ and $\mathcal{I}$ to check if there exists a (s,k,b)-resilient system for $\mathcal{I}$ that satisfies ψ, and to compute such a system whenever it exists. The (s,k)-resilient synthesis problem for ψ and $\mathcal{I}$ is to decide whether for some $b \in \mathbb{N}$, a (s,k,b)-resilient system exists and to find such a system.*

For the brevity of presentation, we only consider the (s,k)-resilient synthesis problem in the following. This choice is motivated by the fact that in practice, glitches often come in bursts, so optimizing towards the maximum possible length of such bursts should have the highest priority.

To simplify the presentation of examples in the following, we will also just list the values that s maps to whenever the assumption list is fixed. For instance, we will write $(\mathit{some}, \mathit{none}, 2)$ instead of $(\{1 \mapsto \mathit{some}, 2 \mapsto \mathit{none}\}, 2)$. For some fixed specification and a fixed interface, we will also call a resilience signature (s,k) *realizable* if there exists a (s,k)-resilient implementation for the specification with the defined interface.

4. SYNTHESIZING ERROR-RESILIENT SYSTEMS

We now show how given a specification $\psi = \psi^a \rightarrow \psi^g$, an interface $\mathcal{I} = (\mathsf{AP}_I, \mathsf{AP}_O)$, a resilience signature s for ψ and a value $k \in \mathbb{N}$, we can perform (s,k)-resilient synthesis. While ψ and $\mathcal{I}$ are necessarily given in any reactive synthesis problem, the latter two describe how error-resilient we want our implementation to be. We show in the next section how to optimize over these, i.e., find the strongest possible values for s and k that are implementable. For the scope of this section, we assume them to be given.

The following construction shows how to modify a GR(1) specification in order to enforce (s,k)-resilience of the implementation along with satisfying the specification. As a consequence, for synthesizing an (s,k)-resilient implementation, we can modify the specification (automatically) and use a standard GR(1) synthesis tool.

For $k \in \mathbb{N}$, we define AP_k to represent a set of signals $\{x_0, \ldots, x_m\}$ such that $m = \lceil \log_2(k+1) \rceil$, and we denote the binary encoding of a number $0 \leq j \leq k$ into AP_k as $(\!|j|\!)$. We also call AP_k the *counter signals*.

DEFINITION 6 (MODIFIED SPECIFICATION). *Given ψ with the set of safety assumptions $\{\psi^a_{s,1}, \ldots, \psi^a_{s,n}\}$, $\mathcal{I}$, s and k, we define the modified interface $\mathcal{I}'$ and the modified specification ψ' as follows:*

$$\begin{aligned} \mathcal{I}' &= (\mathsf{AP}_I, \mathsf{AP}_O \cup \mathsf{AP}_k) \text{ and} \\ \psi' &= (\psi^a_i \wedge \psi'^a_s \wedge \psi'^a_l) \rightarrow (\psi'^g_i \wedge \psi'^g_s \wedge \psi'^g_l) \end{aligned}$$

with

$$\psi'^a_s = \bigwedge_{j \in \{1,\ldots,n\}, s(j)=\mathit{none}} \mathsf{G}\, \phi^a_{s,j} \wedge \bigwedge_{\substack{h \in \{0,\ldots,k\}, \\ D \subseteq \{1,\ldots,n\}, \\ |D|=h+1, \\ \forall j \in D: s(j)=\mathit{some}}} \mathsf{G}\left((\!|h|\!) \rightarrow \bigvee_{j \in D} \phi^a_{s,j} \right),$$

$$\psi'^a_l = \bigwedge_{j \in \{1,\ldots,n'\}} \mathsf{GF}(\neg(\!|k|\!) \vee \phi^a_{l,j})$$

$$\text{for } \psi^a_l = \mathsf{GF}\phi^a_{l,1} \wedge \ldots \wedge \mathsf{GF}\phi^a_{l,n'},$$

$$\psi'^g_i = \psi^g_i \wedge (\!|k|\!),$$

$$\psi'^g_s = \psi^g_s \wedge \bigwedge_{\substack{h \in \{0,\ldots,k\}, \\ D \subseteq \{1,\ldots,n\}, \\ |D| \leq h, \\ \forall j \in D: s(j)=\mathit{some}}} \mathsf{G}\left((\!|h|\!) \wedge \bigwedge_{j \notin D} \phi^a_{s,j} \rightarrow \mathsf{X} \bigvee_{h' \geq h - |D|} (\!|h'|\!) \right),$$

$$\psi'^g_l = \mathsf{GF}\left((\!|k|\!) \vee \bigvee_{j \in \{1,\ldots,n\}} \neg \phi^a_{s,j} \right) \wedge \bigwedge_{j' \in \{1,\ldots,n'\}} \mathsf{GF}\left(\phi^s_{l,j'} \vee \bigvee_{j \in \{1,\ldots,n\}} \neg \phi^a_{s,j} \right)$$

$$\text{for } \psi^g_l = \mathsf{GF}\phi^g_{l,1} \wedge \ldots \wedge \mathsf{GF}\phi^g_{l,n'}.$$

Intuitively, when constructing a modified specification, we add a set of output signals that the system can use to declare how often assumption violations can be tolerated in the near future. The modified formula ψ'^g_i describes that the system should start with a value of k. The progress of the values of AP_k is constrained in ψ'^g_s, which describes that in every step of the system's execution, it can only decrease the value of the counter by the number of assumptions violated. However, the counter value can always be increased. We modify the safety assumptions such that those assumptions that s maps to *some* must only be satisfied if otherwise the counter would drop below 0. As the system can only decrease the counter values with assumption violations, this witnesses that recently k glitches with respect to the *some*-properties have occurred.

The modified liveness guarantees only require the system to satisfy its original liveness guarantees when finitely many assumption violations occur. Also, in that case, the counter must infinitely often have a value of k. While, intuitively, this does not require the existence of a bound $b \in \mathbb{N}$ such that the system is (s,k,b)-resilient, applying a synthesis tool to a specification that we modified according to Definition 6 yields only implementations that have such a bound b. This is because GR(1) synthesis procedures only produce finite-state solutions, for which b is bounded from above by the number of states in the system.

Let us now prove that using the modified specification, we can synthesize (s,k)-resilient systems.

THEOREM 1. *Let $\mathcal{I}$ be an interface, ψ be a generalized reactivity(1) specification, s be a resilience signature, $k \in \mathbb{N}$, and $\mathcal{I}'$ and ψ' be the modified interface and specification constructed from $\mathcal{I}$, ψ, s, and k using Definition 6. We have:*

- *If there exists an (s,k)-resilient system with interface $\mathcal{I}$ for ψ, then there exists a system that satisfies ψ' for the interface $\mathcal{I}'$.*
- *If a system satisfies ψ' for the interface $\mathcal{I}'$, then it is (s,k)-resilient and satisfies ψ.*

PROOF. *Part 1:* Assume that we are given some (s,k)-resilient finite-state machine $\mathcal{M}$. We can augment every state in $\mathcal{M}$ by some counter value that represents the maximum number (in $\mathbb{N} \cup \{\infty\}$) of violations of *some*-assumptions that it is guaranteed to tolerate from the state (along any path). Note that along any transition, the counter can only decrease by as much as the number of assumption glitches witnessed by the transition, as otherwise the counter value

of the predecessor state of a transition would be incorrect. This fact also holds if we cap every number to k. Consider that we augment the output of the system by the additional signals AP_k that it uses to output the current counter value. The set of states and the transitions between the states of the system's finite-state machine are not altered by adding these signals as the counter value only depends on the current state. The modified assumptions $\psi_s'^a$ are engineered to hold along all (s,k)-sane traces then. These are the traces that the system can cope with by the assumption that it is (s,k)-resilient, so it must satisfy ψ_s^g on them. As $\psi_s'^g$ consists of two conjuncts, namely ψ_s^g and the requirement that the counter is updated correctly, it also satisfies $\psi_s'^g$ for (s,k)-sane input streams. Fulfillment of the initialization constraints is trivial.

Since $\mathcal{M}$ is (s,k)-resilient, we must be able to label every state that lies at a loop of states that can be taken infinitely often without violating a safety assumption with k. Otherwise, some input could delay the recovery of the system to a stage at which it can tolerate k more glitches of *some*-properties arbitrarily, so the system would not be (s,k,b)-resilient for any value of b. At the same time, along all loops in the system that can be taken without glitches and while satisfying the liveness assumptions of the system when being taken infinitely often (a "*good*" loop), it must satisfy ψ_l^g when taken infinitely often. Thus, if ψ_l^a is fulfilled and the trace is (s,k)-sane, then $\psi_l'^g$ is fulfilled as then eventually the counter is k and ψ_l^g is fulfilled.

Part 2: Let a finite-state machine with interface $\mathcal{I}'$ be given that satisfies ψ'. We show that this system is (s,k)-resilient for ψ. That is ψ is satisfied for all traces that are (s,k,b)-sane for some $b \in \mathbb{N}$. We set b to be the number of states in the finite-state machine. Note that the counter represented by the AP_k signals can only be decreased by glitches for *some*-assumptions. Furthermore, in between every b steps of the machine's execution without a glitch, the counter has to be reset to k, as otherwise we have found a loop in the execution that can eventually be taken forever and that violates the first conjunct of $\psi_l'^g$. Since continuously having a counter value of less than k also satisfies the liveness assumptions, such a loop would witness the non-satisfaction of ψ' by the system. Since, after at most b steps, we have always found a loop, the claim follows. As the counter value can only be decreased with glitches (by the definition of $\psi_s'^g$), the value always represents an over-approximation of how many glitches still need to be tolerated for a (s,k,b)-sane trace before a b-length recovery phase. The only way to violate ψ_a' is to violate a *none*-assumption or to violate more assumptions than allowed by the counter value in a step. Both cases witness that the trace is not (s,k,b)-sane. Thus, the safety assumptions of ψ' only excluded cases that do not need to be considered for establishing the (s,k)-resilience of a system. On the other hand, if at some point glitches stop occurring, then the system has to satisfy ψ_l^g. Also, ψ_s^g and ψ_i^g are included in the guarantees of ψ'. Therefore, all (s,k,b)-sane traces of the finite-state machine satisfy ψ^g. □

Note that we can just ignore the additional output signals that a system has for a modified specification, so an implementation for a modified specification can also be used for the original specification. Also observe that starting from a GR(1) specification, a modified specification is still in GR(1) form. Thus, we can solve the (s,k)-resilient synthesis problem for a GR(1) specification using a classical GR(1) synthesis procedure by just modifying the specification with the construction of Definition 6.

When applying Definition 6, the number of signals in the specification grows only logarithmically in k, and the number of liveness assumptions and guarantees grows only by 1. Thus, the complexity of a GR(1) synthesis process (which is polynomial in the number of liveness assumptions and guarantees and exponential in the number of signals) grows only polynomially in k. This fact does not contradict the exponential growth of the formula length, as complexity-wise, it is subsumed in the increase of the number of signals.

In Definition 6, we assumed the strict semantics of the implication that connects the assumptions and guarantees in a GR(1) specification. For specifications for the non-strict semantics, one can use the construction from [4] to translate it to an equivalent specification for the strict semantics, and then apply Definition 6. The resulting specification is suitable for one of the many GR(1) synthesis tools with strict implication semantics.

5. FINDING ALL PARETO-OPTIMAL ERROR-RESILIENCE CONFIGURATIONS

In the previous section, we explained how to synthesize (s,k)-resilient systems for given values of s and k. In a formal system development process, we want to automate the task of searching for the best values of s and k in order to ensure that we obtain an as-good-as-possible system. In particular, we want to find all Pareto-optimal solutions. A value pair (s,k) is Pareto-optimal for a specification ψ if there exists no other realizable value pair $(s',k') \neq (s,k)$ that *dominates* (s,k), i.e., such that for all $i \in \{1,\ldots,n\}$, we have $s'(i) \geq s(i)$ (for the total order $none < some < any$) and $k' \geq k$, with the restriction that we treat all values of k as equal if s does not map any assumption to *some* (as then the value of k does not matter). We also call such value pairs (s,k) *resilience configurations* in the following.

As the set of possible resilience configurations (s,k) is infinite, we cannot just enumerate all configurations and check for each of them if an (s,k)-resilient system can be found. Even if we could, such an approach would be inefficient, as we can make use of *monotonicity* – if for some (s,k), we find an implementation, then we can surely find one for a configuration (s',k') that represents weaker resilience requirements to the system.

We propose the following two-step process. We first search for the Pareto-optimal implementable resilience configurations (s,k) for $s : \{1,\ldots,n\} \to \{none, any\}$. Observe that here, the value of k does not matter, so without loss of generality, we can set $k = 0$. In the second step, starting from the realizable resilience configurations found in the first step, we check if we can upgrade some *none*-assumptions to *some*-assumptions and raise the value of k as much as possible. The maximally permissible values of k are always bounded, as otherwise this would witness that we could make all *some*-assumptions to *any*-assumptions, which contradicts the fact that we started from a Pareto-optimal solution for $s : \{1,\ldots,n\} \to \{none, any\}$.

We formalize both steps as a search problem for all maximal elements in a lattice $(P, \leq)$ that are mapped to **true** by some antitone function $f : P \to \mathbb{B}$. In the first step,

Algorithm 1 Pareto-optima finding algorithm

procedure SEARCH($(P, \leq)$,x,succ,f,*found*,*missed*,*done*)
  if $x \in$ *done* **then return** ;
  if $\exists p \in$ *missed* : $p \leq x$ **then return** ;
  if $(\exists p \in$ *found* : $p \geq x) \vee f(x)$ **then**
    for $x' \in \text{succ}(x)$ **do**
      SEARCH(x',succ,f,*found*,*missed*,*done*);
    if $\nexists p \in$ *found* : $x \leq p$ **then**
      found $\leftarrow$ *found* $\cup \{x\}$;
  else
    missed $\leftarrow$ *missed* $\cup \{x\}$;
  done $\leftarrow$ *done* $\cup \{x\}$;
end procedure

P is simply $2^{\{1,\ldots,n\}}$ and $\leq$ is set inclusion. An element of P then contains all assumption indices i that are mapped to *any*. To search for all Pareto-optimal solutions in this context, we can apply algorithms from the field of computational learning, in particular for learning a monotone Boolean function from an oracle. However, such algorithms are typically geared towards minimizing the number of evaluations of f in the context of Boolean functions (see [8] for an overview of classical algorithms), and are not applicable to the second step of our construction, in which a non-Boolean lattice is considered. For simplicity, we use the search algorithm given in Algorithm 1 for both cases, which is suitable for all lattices with a least element. It is based on a solution idea by Gainanov [9] to traverse the lattice from the least element until a Pareto-optimal element is found, but uses a caching scheme to reduce the number of evaluations of f. The first step in our two-step process can be executed through the call

SEARCH($(P_1, \leq_1)$, $\emptyset$, succ_1, f_1, $found_1$, $missed_1$, $done_1$)

where the parameters are set as

$found_1 \leftarrow \emptyset$;
$missed_1 \leftarrow \emptyset$;
$done_1 \leftarrow \emptyset$;
$(P_1, \leq_1) \leftarrow (2^{\{1,\ldots,m\}}, \subseteq)$.

For reasons of readability, the functions f_1 and $succ_1$ are given in Algorithm 2. The function REALIZABLE(s, k, ψ) denotes checking (s, k)-resilient realizability of the specification ψ using Definition 6 and a GR(1) synthesis procedure. The parameters to SEARCH are passed by reference, so that the contents of $found_1$, $missed_1$, and $done_1$ are retained during the recursive evaluation of SEARCH and can be used for caching the results already computed.

After we have identified the realizable resilience configurations that are restricted to *none* and *any* entries, we turn towards the second step of the construction, in which we search for the remaining ones. We apply the SEARCH procedure

SEARCH($(P_2, \leq_2)$, $(\{\{1, \ldots, n\} \mapsto none\}, 1)$, succ_2, REALIZABLE, $found_2$, $missed_2$, $done_2$)

with the following parameters:

$found_2 \leftarrow \text{conv}(found_1)$;
$missed_2 \leftarrow \text{conv}(missed_1)$;
$done_2 \leftarrow \emptyset$;
$(P_2, \leq_2) \leftarrow (\{1, \ldots, n\} \to \{any, some, none\}) \times \mathbb{N}_{\geq 1} \setminus (\{1, \ldots, n\} \to \{any, none\}) \times \mathbb{N}_{\geq 2}, \text{cmp}_2)$.

Algorithm 2 Lattice successor computation function and antitone function definition for step 1 of the Pareto-optimal solution finding procedure

function $\text{f}_1(x)$
  return REALIZABLE($\{i \mapsto any \mid i \in \{1, \ldots, n\}, i \in x\} \cup \{i \mapsto none \mid i \in \{1, \ldots, n\}, i \notin x\}, 0, \psi$);
end function

function $\text{succ}_1(A)$
  return $\{A \cup \{x\} \mid x \in \{1, \ldots, n\}\} \setminus \{A\}$;
end function

Here, the function *conv* translates a resilience signature of the shape used in step 1 to a resilience configuration for step 2, i.e., for every $x \subseteq \{1, \ldots, n\}$, we have

$$\text{conv}(x) = \{(\{i \mapsto \tau(x(i)) \mid i \in \{1, \ldots, n\}\}, 1) \mid x \in found_1\}$$

for $\tau(x) = any$ if $x =$ **true** and $\tau(x) = none$ if $x =$ **false**.

The comparison operator cmp_2 and the function succ_2 for computing the direct successors of an element in the lattice $(P_2, \leq_2)$ are given in Algorithm 3. Note that the succ_2 function computes *any* direct successors of a resilience configuration, but we have a slightly more complicated definition of the lattice in step two than in step one. It excludes all resilience configurations (s, k) that are dominated by the ones found to be realizable in step 1 (i.e., are contained in $found_1$) and for which $k > 1$. As we know that in such cases, for any value of k, our specification is (s, k)-resiliently realizable, we prevent to needlessly search for larger values of k in this way. Note that any resilience configuration in the lattice is still reachable by succ_2 steps from the least element $(\{\{1, \ldots, n\} \mapsto none\}, 1)$, so we do not miss any solutions in this way. However, it is ensured that our search only needs finite time due to this modification since only finitely many reachable resilience configurations are mapped to **true** by REALIZABLE. The reason is that for cases (s, k) not covered by the ones from step one, there is a maximum number for k such that the specification is (s, k)-realizable, but not $(s, k + 1)$-realizable. To see this, assume that there exists a value of s such that the specification is (s, k)-resiliently realizable for any value of k. Then it is also realizable for $(s', 0)$ for which s' results from setting all $i \in \{1, \ldots, n\}$ to *any* if they are set to *some* or *any* in s. Such configurations would have been found in step 1 however, and as a consequence, $succ_2$ would have never computed (s, k) as a possible successor of any resilience configuration for $k > 1$.

To speed up the computation in step 2, $found_2$ and $missed_2$ are initialized with information stored into $found_1$ and $missed_1$ after the first step. Setting $k = 1$ for entries added to $found_2$ is motivated by the fact that in $(P_2, \leq_2)$, a value of $k = 1$ can only be exceeded for resilience configurations (s, k) for which s is not dominated by some resilience configuration that is found to be realizable during the first step of the overall search procedure, which would not be covered by a case stored in $found_1$ anyway. Note that some of the configurations by which we initialize $found_2$ may be Pareto-optimal, whereas others might not. Thus, after the call to SEARCH, the set $found_2$ can contain non-Pareto-optimal configurations. Whenever this is a concern, we can post-process the set $found_2$ after the call to SEARCH by simply removing resilience configurations that are dominated by other configurations in $found_2$.

Algorithm 3 Lattice navigation functions for step 2 of the Pareto-optimal solution finding procedure

function $\text{cmp}_2((s,k),(s',k'))$
  return $(\forall i \in \{1,\ldots,n\} : s(k) \leq s'(k)) \wedge (k \leq k')$;
end function

function $\text{succ}_2(s,k)$
  $X \leftarrow \{(s',k) \mid s' \neq s, \exists i \in \{1,\ldots,n\} : \forall j \in \{1,\ldots,n\} \setminus \{i\} : s'(j) = s(j), (s(i) = \mathit{none} \wedge s'(i) = \mathit{some}) \vee (s(i) = \mathit{some} \wedge s'(i) = \mathit{any})\}$;
  if $\exists i \in \{1,\ldots,n\} : f(i) = \mathit{some} \wedge \forall f \in \mathit{found}_1,\ \exists i \in \{1,\ldots,n\} : \neg f(i) \wedge (s(i) \neq \mathit{none})$ **then**
    $X \leftarrow X \cup \{(s,k+1)\}$;
  return X
end function

To illustrate the search algorithm, Figure 2 depicts the call graph of the SEARCH function for a specification with two assumptions for the set of Pareto optima $\{(\mathit{any},\mathit{none},1),(\mathit{none},\mathit{any},1),(\mathit{some},\mathit{some},2)\}$. Step one yields $\{(\textbf{true},\textbf{false}),(\textbf{false},\textbf{true})\}$ in this case, so that found_2 is initialized with $\{(\mathit{any},\mathit{none},1),(\mathit{none},\mathit{any},1)\}$.

The graph represents the candidate resilience configurations during the recursive evaluation of the SEARCH procedure. Every configuration is labeled by whether it is realizable ($^+$) or not ($^-$). The return edges are dotted, whereas calls that cause the SEARCH function to return in the first two lines due to caching in the sets missed_2 and done_2 are dashed. All other edges represent that the third line of Algorithm 1 is reached and thus the function f given to the algorithm is evaluated or a resilience configuration is covered by one of found_2 already. The latter is the case for the edges 1, 2, 21, and 24 in the figure. Thus, the GR(1) synthesis procedure needs to be called six times in this example.

It can be seen that the SEARCH algorithm mitigates the problem that a lattice element can be reached from multiple successors by caching whether an element has already been considered in *done*. This happens at the edges 7, 22, and 25. For edge 12, the unrealizability of the configuration $(\mathit{any},\mathit{some},2)$ is deduced from the fact that before return edge 4, $(\mathit{any},\mathit{some},1)$ is stored in missed_2.

In terms of complexity, the number of calls to the REALIZABLE function in both steps of the algorithm is bounded from above by $2(n+1)\cdot \#\mathit{Sig}$, where $\#\mathit{Sig}$ is the number of realizable resilience configurations (s,k) in P_2 for which k is not needlessly ≥ 1, i.e., for which there is no realizable resilience configuration $(s',1)$ that dominates (s,k).

6. EXAMPLES AND EXPERIMENTS

We implemented the constructions from Section 4 and Section 5 in a Python script that interfaces the GR(1) Synthesis tool `slugs` [6], using strict implication semantics. We consider three benchmarks to evaluate the practical applicability of our resilient synthesis approach, which we describe in the following. All computation times are given for an Intel Xeon 2.40GHz computer running Linux, with a 4 GB memory limit, which was never exceeded. All computations are performed single-threaded. During the computation of the realizable resilience configurations, only realizability is checked, but no implementation is computed. All implementation sizes are given in states for explicit-state implementations that were computed in additional runs of `slugs`.

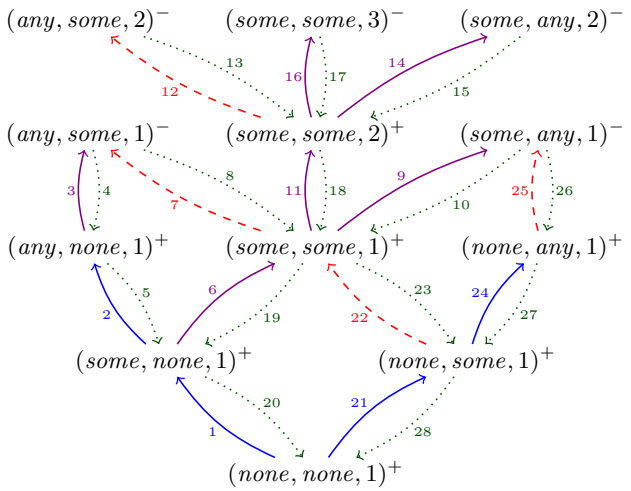

Figure 2: Graphical representation of the call structure during step 2 of the search algorithm for some specification with two assumptions.

6.1 Water Reservoir Level

As a simple introductory example, we consider a water reservoir controller. The reservoir is required to always hold between 10 and 99 gallons of water. There are two in-flows that are uncontrollable for the system. Each of them either delivers no water at all or two gallons of water at every time step. The controller can decide to release three gallons of water in a time step. We start with a reservoir level of 10 gallons.

By default, this setting is unrealizable as the controller cannot avoid the reservoir to overflow. This can be seen from the fact that in every step, four gallons of water can be added, but only three gallons can be released. With the two assumptions for each of the in-flows that they can only release water in every second time step, the setting becomes realizable.

Applying the constructions from the previous sections yields $\{(\mathit{any},\mathit{some},87),(\mathit{some},\mathit{any},87),(\mathit{some},\mathit{some},175)\}$ as the set of Pareto-optimal realizable resilience configurations. Obtaining this set takes 4 minutes and 52 seconds of computation time (including 270 calls to the `slugs` synthesis tool). For comparison, synthesizing a non-error-resilient controller for the water reservoir controller takes 0.35 seconds. Checking the realizability of a modified specification takes 1.07 seconds in the mean, and the number of (reachable) states of the computed implementations increases from 2043 for the non-resilient implementation to 2383 for the Pareto optima with $k = 87$, and 2489 for the other optimum.

The structure of the set of Pareto-optimal solutions can be explained from the fact that we need glitches for both of the in-flows in order to reach a situation in which the 3 gallons/time unit of the out-flow are insufficient to cope with the in-flows. Due to the difference of one gallon per time unit between the maximum in-flow and the maximum out-flow one would expect $(\mathit{any},\mathit{some},89)$ to be a realizable resilience configuration, but in fact it is not. The reason is that the environment can initially trigger only a single in-flow, which yields a reservoir level of 12 gallons, which the controller cannot reduce without violating its specification. Starting

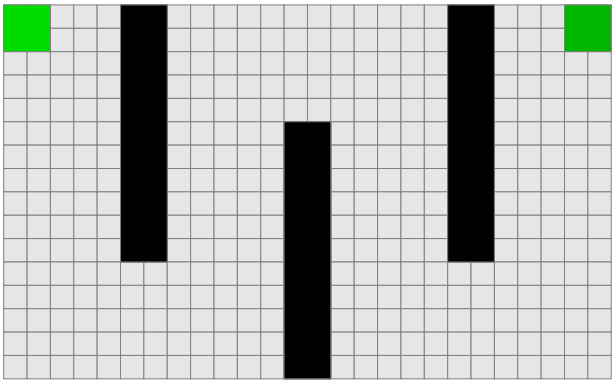

Figure 3: Workspace of the robot for the scenario of Section 6.2.

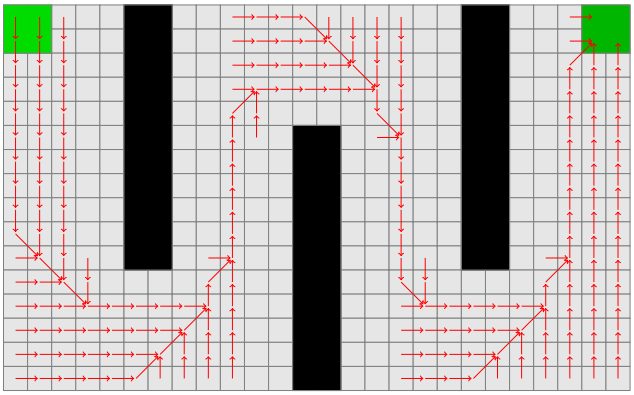

Figure 4: Robot behavior in the absence of glitches.

from 12 gallons, 87 cycles with two glitches in every cycle are then needed for the environment to enforce exceeding the 99 maximally allowed gallons in the reservoir. If one of the assumptions is an *any*-assumption, only one of the glitches counts towards k, leading to a maximum value for k of 87. If both glitches count, we get a maximally possible value of $k = 175$. The difference of one is explained by the fact that a computation cycle with a single glitch is never a problem for the system, as this means that our in-flow is at most 3 gallons, which can be handled. Thus, one additional glitch is permitted.

6.2 Robot Motion Planning

As a second example, we consider a problem in which a robot is required to deliver a package from the top left corner of a workspace (shown in Figure 3) to the top right corner of a workspace whenever it is triggered by some input signal. The workspace features some obstacles that the robot must not collide with. Running into the boundaries of the workspace is allowed. The robot has a sensor for its current position and can trigger motion into eight directions (north west, north, north east, etc.). Let us call the directions along the width and height as x and y direction, respectively.

We have three safety assumptions in this setting: (1) the robot position cannot change by more than one cell (horizontally, vertically, or diagonally) per execution step, (2) the x-position of the robot is updated according to whether it moves westwards or eastwards, and (3) the y-position of the robot is updated according to whether it moves northwards or southwards.

Finding all Pareto-optimal resilience configurations $\{(any, none, none, 1), (some, some, some, 1), (none, some, any, 1), (none, some, some, 2)\}$ takes 16.8 seconds (including the 22 calls to `slugs`). As assumptions (2) and (3) together imply assumption (1), the first two of these configurations are not interesting. The third configuration allows arbitrarily many glitches in the y-direction, but only a single one in the x-direction. The asymmetry between the x and y directions is due to that there are a columns along the y direction in the workspace without obstacles. Whenever we get a sequence of glitches for assumption (3), the robot can move to these columns to avoid bumping into obstacles, and later return to fulfilling its delivery task. The last of the resilience configurations is the one in which the robot maximizes its resilience against small dislocations. Since the resulting implementation has more reachable workspace cells, its size increases from 164 states of the non-resilient implementation to 7136 states. Figure 4 shows the paths taken by the robot for getting the package to the top right corner in case that no glitch occurred so far for the synthesized implementation with resilience configuration $(none, some, some, 2)$. In this figure, the arrows show the transitions that the robot shall take if it ends up in a cell from which an arrow originates. The robot does not reach the cells without outgoing arrows under the corresponding strategy. It can be seen that the robot keeps a good distance to the obstacles. In case of dislocations of the robot, it performs different actions. Figure 5 shows the actions of the robot for the case that recently two glitches have been observed, but no delivery is to be made, i.e., the only task of the robot at the respective point in time is to avoid obstacles.

6.3 Robot Motion Planning with Moving Obstacles

We now consider a variant of the motion planning problem of Section 6.2 with a moving obstacle introduced to the workspace shown in Figure 6. The robot to be controlled starts in the upper left corner and has the assumptions and guarantees described in Section 6.2. The controller also obtains as input the position of the moving obstacle that has a width of two cells and an equal height. In addition to the three assumptions about the robot position updates, we have two additional ones: (4) that the obstacle can only move in every second execution step, and (5) that the obstacle can only move to adjacent cells in every step while never touching the fixed obstacles (black) and the gray boundary cells, which can only be crossed by the robot to be controlled. Assumption (4) is necessary to prevent the obstacle from moving into the path of the robot all of the time, while (5) ensures that the obstacle cannot jump onto the robot and can also not block the pickup and delivery regions, which are colored in Figure 6 in the same way as in Figure 3.

Analyzing the scenario takes 361 minutes (with 75 calls to `slugs` for modified specifications and an average computation time of 288.5 seconds per call). The set of Pareto optima is $A = \{(any, none, none, some, none, 47)\}$, and synthesis without error-resilience for the original specification takes 9 minutes and 23 seconds. The number of states in the implementation increases from 133819 to 1587114 for the resilience signature in A. As again the satisfaction of

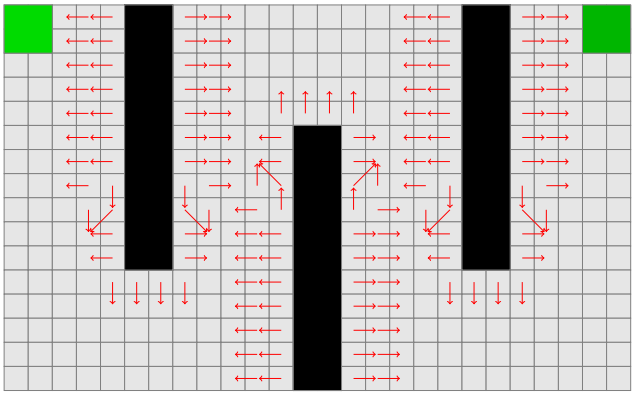

Figure 5: Robot backup behavior after two glitches.

assumptions (2) and (3) together imply the satisfaction of assumption (1), the singular set of Pareto optima shows that the only assumption for which we can tolerate glitches is the one that disallows the obstacle to move at the same speed as the robot (i.e., one cell per cycle for both x and y directions). However, we can tolerate bursts of glitches of length 47 for this assumption. Intuitively, the space available for the robot to pass by the obstacle is not sufficient to tolerate a glitch in its motion, but the obstacle being too fast a few times can be compensated.

We also considered a moving obstacle in a 16×16 grid without fixed obstacles in which the only task for the robot is to avoid colliding with the moving obstacle. The assumptions are the same as in the previous case. After 4 minutes and 29 seconds (90 calls to `slugs`), we obtain the following Pareto-optimal resilience configurations:

1. $(any, none, none, some, none, 20)$
2. $(some, some, some, some, none, 1)$
3. $(none, some, some, some, none, 2)$
4. $(none, some, none, some, none, 3)$
5. $(none, none, some, some, none, 3)$
6. $(none, some, none, none, none, 5)$
7. $(none, none, some, none, none, 5)$

Due to the smaller workspace, only 20 glitches of the assumption that the obstacle must not move too quickly can be tolerated (if all other assumptions are strict). Again, due to the implication between assumptions (2), (3), and (1), the second resilience configuration can be ignored. This time, robot motion glitches and speed glitches for the moving obstacle can be tolerated in combination, namely 2 glitches in total if the direction of the robot motion glitch is not fixed, and 3 glitches otherwise. If only robot motion glitches in x or y direction need to be tolerated, then 5 glitches are admissible. If both of them can occur, then only 2 glitches can be tolerated (which is a special case of resilience configuration number 3).

7. RELATED WORK

Reasoning about the changes (or preservations) of system's properties under uncertainties and disturbances has been considered not only in the control theory literature but also in synthesis of discrete reactive controllers. Numerous notions of robustness (and sensitivity) have been defined for different applications. For example, Tabuada et al. [15] consider the problem of synthesizing a program whose behavior in case of disturbances stays close to the nominal-case behavior and at the same time the effect of disturbances vanishes over time. It can be seen that this robustness criterion is motivated by similar notions in controls [18, 16] and heavily depends on the existence of a distance notion between different input/output streams, in contrast to the error-resilience criterion that we use in this paper. Additional work includes [3] which focuses on safety properties and uses the ratio of the distances between allowed and observed system behavior to that of the environment behavior as a measure of sensitivity. Reference [17] on the other hand synthesizes reactive controllers that are robust to uncertainties in the underlying open finite transition systems (e.g., due to unmodeled transitions).

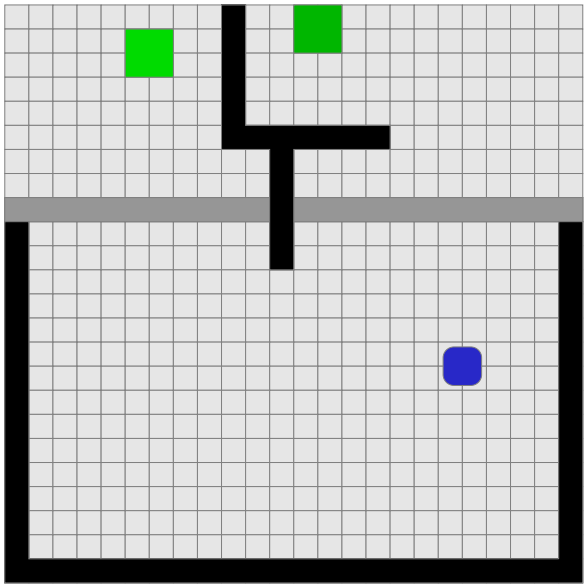

Figure 6: Robot workspace with a moving obstacle.

Ehlers [5] discusses the synthesis of systems that are robust in the sense that after safety assumption violations, the system must return to an operation mode in which the safety guarantees are satisfied again after only finitely many violations of these. Thus, such systems have a backup strategy for anomalous operating conditions without the need for the system engineer to specify one. Bloem et al. [2] consider a strengthened version of the approach in which a robust system is required to satisfy its liveness guarantees even in the presence of infinitely many safety assumption violations. In the approaches of [5] and [2], the robustness criteria are purely qualitative. In contrast to these works, in our new approach, all of the guarantees should always be satisfied without deviation. We systematically extend the set of environment behaviors that can be dealt with by the system synthesized. Our error-resilience definition is thus closer to the classical notions of robustness in control, in which the admissible environment conditions of the computed controllers are to be loosened as much as possible.

8. CONCLUSION & DISCUSSION

In this paper, we considered the problem of synthesizing systems that are resilient against environment assumption

violations. The key contribution is a construction to change a generalized reactivity(1) specification to one of the same type, but that encodes the search for an implementation that exhibits the desired error-resilience level. This encoding allows to use off-the-shelf generalized reactivity(1) synthesis tools for obtaining error-resilient systems. Our construction distinguishes between two types of error-resilience for every assumption: resilience against arbitrarily long sequence of glitches, and resilience against at most k glitches (for some k). To find all Pareto-optimal assignments of the assumptions to these types and to maximize the value of k, we described an algorithm to search for these Pareto optima and thus provided a fully-automated method to explore how resilient an implementation for a specification can be made.

Our contribution not only allows to synthesize error-resilient implementations, but also helps a system engineer with the manual construction of error-resilient systems. Starting with a specification for the system to be constructed and a corresponding implementation, we can analyze whether the implementation is already optimal with respect to error-resilience by comparing the error-resilience level of the implementation against the Pareto optima found in the synthesis process.

The definition of the synthesis problem for error-resilient systems that we considered in this paper has many conceivable variants that we did not discuss in detail. For example, we could require that the length of a recovery period (referred to as the value of the variable b in Section 3) shall be minimized. We can approximate this behavior by changing the definition of $\psi_l'^g$ in our specification modification construction (by replacing all liveness guarantees $\mathsf{GF}\phi$ in $\psi_l'^g$ by $\mathsf{GF}((\langle\!|k|\!\rangle \wedge \phi) \vee \bigvee_{i=0}^{k-1} \langle\!|i|\!\rangle \wedge \neg\mathsf{X}\langle\!|i|\!\rangle)$ in order to count changing the counter value as progress whenever it is not yet k). However, a slight modification of the GR(1) synthesis tool is advisable in order to let the implementation favor transitions that make progress towards the unmodified liveness guarantees whenever possible while working towards increasing the counter value.

Another variant of practical relevance is strengthening the definition of error-resilience to require the satisfaction of a specification's liveness guarantees also in the case of infinitely many safety assumption violations. Note that this variant only makes sense if the resilience signature does not map some assumptions to *any*, as otherwise the implementability of a system for this error-resilience definition would witness that the *any*-assumptions are actually not needed. Synthesis for such a variant of the error-resilience definition is best performed by using a modified synthesis algorithm, in which we use the unmodified liveness assumptions and guarantees, but at the same time force the system to always eventually set the counter to a value closer to k or k itself, regardless of the satisfaction of the liveness assumptions.

These two variants of the methods proposed in this paper retain the same complexity of the synthesis process.

Acknowledgements

The first author was partially supported by the ERC under grant agreement no. 259267 and NSF ExCAPE CCF-1139025/1139138. The second author was partially supported by the AFOSR (FA9550-12-1-0302) and ONR (N00014-13-1-0778).

9. REFERENCES

[1] S. Almagor, U. Boker, and O. Kupferman. Formalizing and reasoning about quality. In *ICALP (2)*, pages 15–27, 2013.

[2] R. Bloem, H.-J. Gamauf, G. Hofferek, B. Könighofer, and R. Könighofer. Synthesizing robust systems with RATSY. In *SYNT*, volume 84 of *EPTCS*, pages 47–53, 2012.

[3] R. Bloem, K. Greimel, T. A. Henzinger, and B. Jobstmann. Synthesizing robust systems. In *FMCAD*, pages 85–92, 2009.

[4] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

[5] R. Ehlers. Generalized Rabin(1) synthesis with applications to robust system synthesis. In *NASA Formal Methods*, pages 101–115, 2011.

[6] R. Ehlers, C. Finucane, and V. Raman. Slugs GR(1) synthesizer. http://github.com/ltlmop/slugs, 2013.

[7] R. Ehlers, R. Könighofer, and G. Hofferek. Symbolically synthesizing small circuits. In G. Cabodi and S. Singh, editors, *FMCAD*, pages 91–100. IEEE, 2012.

[8] T. Eiter, K. Makino, and G. Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008.

[9] D. Gainanov. On one criterion of the optimality of an algorithm for evaluating monotonic boolean functions. *USSR Computational Mathematics and Mathematical Physics*, 24(4):176–181, 1984.

[10] C.-H. Huang, D. Peled, S. Schewe, and F. Wang. Rapid recovery for systems with scarce faults. In M. Faella and A. Murano, editors, *GandALF*, volume 96 of *EPTCS*, pages 15–28, 2012.

[11] G. Jing, R. Ehlers, and H. Kress-Gazit. Shortcut through an evil door: Optimality of correct-by-construction controllers in adversarial environments. In *IROS*, pages 4796–4802. IEEE, 2013.

[12] S. C. Livingston. gr1c GR(1) synthesizer. http://github.com/slivingston/gr1c, 2013.

[13] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[14] V. Raman, N. Piterman, and H. Kress-Gazit. Provably correct continuous control for high-level robot behaviors with actions of arbitrary execution durations. In *ICRA*. IEEE, 2013.

[15] P. Tabuada, A. Balkan, S. Y. Caliskan, Y. Shoukry, and R. Majumdar. Input-output robustness for discrete systems. In *EMSOFT*, pages 217–226, 2012.

[16] D. C. Tarraf, A. Megretski, and M. A. Dahleh. A framework for robust stability of systems over finite alphabets. *IEEE Transactions on Automatic Control*, 53(5):1133–1146, 2008.

[17] U. Topcu, N. Ozay, J. Liu, and R. M. Murray. On synthesizing robust discrete controllers under modeling uncertainty. In *Hybrid System: Computation and Control*, pages 85–94, 2012.

[18] J. C. Willems. Dissipative dynamical systems part i: General theory. *Archive for rational mechanics and analysis*, 45(5):321–351, 1972.

APPENDIX

We provide some additional information here that may be of interest to a few readers.

A. A NOTE ON THE SIZES OF THE IMPLEMENTATIONS

In the main part of the paper, we provided some statistics on the sizes of the computed implementations. These were provided as the numbers of states in explicit-state implementations. The synthesis tool `slugs` that we used for our experiments computes these as Mealy-machines in which the states are labeled by the last input to the system, the last output to the system, and the number of the liveness guarantees for which the implementation is trying to make progress when being in that state.

For some benchmarks, no such explicit-state implementation sizes were given. This is because explicit-state implementations are not the best target for larger specifications. In fact, GR(1) synthesis tools such as `slugs` use *binary decision diagrams* (BDDs) as symbolic data structure to speed up solving the *synthesis game* that is internally built for solving the synthesis problem. The resulting strategy in the game, which represents an implementation, can then be given as a list of BDDs. With a simple wrapper, they are usable as a software controller. Alternatively, a symbolic *strategy extraction* technique such as the one in [7] can be used to compute a small circuit from the BDDs.

The BDD node counts of the computed implementations in the examples in the main part of the paper were:

- Water reservoir controller:
  - Base version: 328
  - Resilience signature $(some, some, 175)$: 7098
  - Resilience signature $(any, some, 87)$: 3305
  - Resilience signature $(some, any, 87)$: 3252
- Robot motion planning without obstacles:
  - Base version: 678
  - Resilience signature $(none, none, some, 4)$: 2545
  - Resilience signature $(none, some, some, 2)$: 13459
  - Resilience signature $(some, some, some, 1)$: 9927
  - Resilience signature $(any, none, none, 1)$: 4797
- Robot motion planning with moving obstacle and delivery task:
  - Base version: 41793
  - Resilience signature $(any, none, none, some, none, 47)$: 666761
- Robot motion planning with moving obstacle and without delivery task:
  - Base version: 309
  - Resilience signature $(any, none, none, some, none, 20)$: 74072
  - Resilience signature $(some, some, some, some, none, 1)$: 7129
  - Resilience signature $(none, some, some, some, none, 2)$: 13390
  - Resilience signature $(none, some, none, some, none, 3)$: 13665
  - Resilience signature $(none, none, some, some, none, 3)$: 37702
  - Resilience signature $(none, some, none, none, none, 5)$: 12753
  - Resilience signature $(none, none, some, none, none, 5)$: 49708

In the main part of the paper, we preferred to give the numbers for explicit-state implementations as the meanings of the BDD node counts are hard to interpret and depend on the *variable reordering heuristic* used in the BDD library.

B. A NOTE ON THE INPUT COMPLEXITY OF THE APPROACH

In the main part of the paper, we discussed the *output complexity* of our algorithm for finding optimally error-resilient implementations, i.e., how much harder the synthesis problem gets when measured in the quality of the *output*. This view is suitable for the practice of synthesizing error-resilient systems, as a longer computation time is acceptable when obtaining an implementation with a good level of error-resilience.

The traditional approach in complexity theory is however to examine the *input complexity* of a problem or algorithm, i.e., how much computation time or space is needed in order to perform the task. For the sake of completeness, let us now have a look.

The time complexity of the standard GR(1) synthesis problem is:

- linear-time for a fixed set of variables (by having a big lookup table for all possible specifications and reading the input to select the right entry[1])
- linear-time for a fixed specification (by having a big lookup table for all possible variable set partitionings into input and output variables and reading the input to the synthesis problem to select the right entry)
- exponential-time when counting both the variable set and the specification as part of the input

A standard implementation of the GR(1) synthesis algorithm runs in time $O((2^v)^4 \cdot n_l^a \cdot n_s^a)$, where v is the number of input and output propositions, n_l^a is the number of liveness assumptions, and n_l^a is the number of liveness guarantees. Both n_l^a and n_l^a are set to 1 if there are no properties of the respective types in the specification. Thus, the algorithm runs in time polynomial in the number of liveness properties, but exponential in v.

Applying the construction from Definition 6 before using the GR(1) synthesis procedure leads to a blowup of the specification: for some value of k, the specification length growths exponentially in k, although the specification size, i.e., the number of distinct subformulas appearing in specification parts, grows only polynomially when applying Definition 6. A tight integration of the specification modification

[1]Note that in GR(1) specifications, one can write Boolean formulas only over the variable values in the current computation step and the next computation step. Therefore, there are only finitely many *semantically* different formulas that can be initialization, safety, or liveness assumptions or guarantees in GR(1) specifications, which make the set of specifications over a fixed set of variables *finite*. During reading the specification, all that needs to be done is to parse the specification parts according to which possible value table over the variables in the current and next state they correspond to.

step and the synthesis algorithm can thus prevent the exponential blowup. At the same time, the game blows up by a factor that is only polynomial in the value of k.

Note that the highest value of k that needs to be considered is $2^v \cdot n_s^a$, where n_s^a is the number of safety assumptions in the specification. This is because 2^v is the number of positions in the non-error-resilient synthesis game. From any position in the game, it can take the environment player at most 2^v steps with safety assumption violations to enforce a safety guarantee violation as otherwise, a loop along a path to the guarantee violation could be witnessed, which could be cut out. As all values of k greater than $2^v \cdot n_s^a$ thus witness that arbitrarily many assumption violations can be tolerated by an implementation (in every step, there can be up to n_s^a violations at the same time), and this case would be found by step one of our overall Pareto optima search algorithm, it follows that considering only values of k up to a limit of $2^v \cdot n_s^a$ suffices.

As the synthesis game can thus only blow up by a factor that is polynomial in $k = 2^v \cdot n_s^a$, and the number of liveness assumptions and guarantees only increases by at most 1 when applying Definition 6, it follows that all specification analysis steps in the Algorithm given in Section 5 have a complexity that is exponential in v and polynomial in the length/size of the specification.

The number of these steps is now bounded by $2^v \cdot n_s^a \cdot 3^{O(n_s^a)}$, as this is the number of different resilience signatures with a bound $\leq 2^v \cdot n_s^a$. This leads to an overall complexity that is exponential in the number of atomic propositions of the original specification and polynomial in the length/size of the specification, except for the number of safety assumptions, in which it is exponential.

Overall, the combined input (time) complexity of the algorithm is in EXPTIME, and the linear-time arguments for the cases of fixing the set of variables or fixing the specification still apply.

Note that in practice, synthesis tools are used that are based on *binary decision diagrams*, which employ a number of heuristics that make it virtually impossible to give correct upper bounds on computation times that are not ridiculously conservative.

C. ON A VARIANT OF THE ERROR-RESILIENT SYNTHESIS PROBLEM

In the conclusion of the main part of the paper, we give an outlook on possible variants of the error-resilient system synthesis problem. In the second of these variants, we require that the system's liveness guarantees must also be satisfied when infinitely many glitches occur.

As stated in the conclusion, this does not alter the complexity of the synthesis problem (it is still EXPTIME in the number of atomic propositions in the original specification and in the binary representation of k). However, for a sound implementation of a synthesis algorithm for this modified error-resilience notion, the solution algorithm needs to be altered to support *two-pair Street games*. Bloem et al. [2] give a description of how to solve such games.