Resilient Abstraction-Based Controller Design

Stanly Samuel, Kaushik Mallik, Anne-Kathrin Schmuck, and Daniel Neider

***Abstract*— We consider the computation of resilient controllers for perturbed non-linear dynamical systems w.r.t. linear-time temporal logic specifications. We address this problem through the paradigm of Abstraction-Based Controller Design (ABCD) where a finite state abstraction of the perturbed system dynamics is constructed and utilized for controller synthesis. In this context, our contribution is twofold: (I) We construct abstractions which model the impact of occasional high disturbance spikes on the system via so called disturbance edges. (II) We show that the application of resilient reactive synthesis techniques to these abstract models results in closed loop systems which are optimally resilient to these occasional high disturbance spikes. We have implemented this resilient ABCD workflow on top of SCOTS and showcase our method through multiple robot planning examples.**

I. Introduction

With the advent of digital controllers being increasingly used to control safety-critical cyber-physical systems (CPS), there is a growing need to provide formal correctness guarantees of such controllers. A recent approach to achieve this goal is the so-called *Abstraction-Based Controller Design* (ABCD) [2], [22]. ABCD is usually performed in three-steps. First, a finite state abstraction is computed from a given non-linear continuous dynamical system by discretizing the state and input space. Second, given this abstraction and a linear-time temporal logic specification, a discrete control strategy is synthesized. Finally, the discrete control strategy is refined to a continuous controller for the given system, which serves as the output of the procedure. ABCD has been implemented in various tools for a variety of classes of systems and specifications, with numerous improvements over the basic procedure, e.g., [16], [20], [10], [11], [12].

An important feature of controllers for CPS is their robustness against modelling uncertainties and unforeseen operating conditions. The ABCD workflow therefore has a build-in robutstification step; given a uniform upper bound W on the uncertainty of continuous trajectories, the abstract transition system over-approximates all possible trajectories of the original system.

This approach, however, has one important disadvantage. Consider for example a mobile robot serving coffee in an office building. As it needs to get to the kitchen to get coffee, it sometimes passes through water spilled on the floor which results in an imprecise motion actuation. In this case, increasing W to contain all possible disturbances caused by spilled water would result in a very conservative abstract model which renders the resulting abstract controller synthesis problem (e.g., always eventually serving a requested coffee) unrealizable. A more optimistic choice of W allows to design a controller but sacrifices rigorous correctness guarantees – if water is spilled the specification might be violated, e.g., the robot might bump into a door frame. Intuitively we however know that it is very unlikely that water is spilled *everywhere* and *all the time*, which would indeed make the specification unrealizable. Further, knowing that there might be spilled water, we would like the robot motion controller to be "risk-aware", i.e., to avoid going close to the door frames.

Contribution. In our work, we automatically synthesize a controller which is correct w.r.t. a nominal disturbance W_{normal} (e.g., no water spilled on the floor), and in addition, is "risk-minimizing" w.r.t. larger disturbance spikes in $W_{\mathrm{high}} \supset W_{\mathrm{normal}}$ that may occur any time.

Formally, we build on top of the basic three step procedure of ABCD. First, we obtain a *risk-aware* finite state abstraction for the given continuous dynamical system by adapting the framework of feedback-refinement relations [18]. Second, we compute a maximally-resilient control strategy for this abstraction by an adaptation of the two-player game algorithm to compute *Optimally Resilient Strategies* as defined by Neider et.al. [5], [17]. Finally, the control strategy so obtained can be refined to a continuous controller for the original system, while retaining the correctness guarantees and the resilience of the states. This three step procedure is what we call the *Resilient Abstraction-Based Controller Design*.

Related Work. Ensuring robustness of discrete, *event-based* control strategies is an active field of research in reactive synthesis. Within *resilient ABCD*, we employ the

S. Samuel is with IISC, Bengaluru, India. Email: stanly@iisc.ac.in.

K. Mallik, D. Neider, and A.-K. Schmuck are with MPI-SWS, Kaiserslautern, Germany. Email: {kmallik, neider, akschmuck}@mpi-sws.org.

method introduced in [5], [17], where *disturbance edges* are utilized to formalize resilience and are assumed to be given. Unfortunately, a good disturbance model is not always easy to obtain in practice. Our *resilient ABCD* method, however, *automatically* injects disturbance edges (i.e., transitions induced by continuous trajectories disturbed by a "spike" from W_{high}) into the abstract model.

Another approach to robust reactive synthesis considers particular specifications of the form $A \rightarrow G$; in every environment that fulfills the assumption A the controller needs to enforce the property G. In this context robustness is for example understood w.r.t. assumption violations[4], [7], [9], hidden or missing inputs [3] or unexpected jumps in the game graph [6]. While the abstraction generated within our *resilient ABCD* method can be understood as a particular safety assumption A for the given synthesis problem interpreted as G, the assumption violations we consider are already explicitly modeled by disturbance edges which are *automatically generated*. This avoids analyzing assumptions for possible faults and allows for a more fine-grained analysis assigning resilience values to states rather than whole systems.

While lifting the synthesis of robust controllers to the abstract domain, *resilient ABCD* still retains the intuitive local nature of *robustness* of continuous systems by injecting disturbance edges only locally. This is closely related to work in robust CPS design, where continuous and discrete metrics are imposed to construct abstractions that allow for robust controller synthesis [19], [23], [14]. In this context, robustness is mostly understood as ensuring that the controlled system remains "close" to a chosen execution path despite disturbances. In contrast, *resilient ABCD* exploits the existence of multiple different control strategies and picks the most resilient among them. This optimizes resilience over the infinite time horizon of the system's execution.

For particular classes of continuous-time systems and temporal-logic specifications, controllers can be designed without explicitly constructing an abstraction by state-space gridding. In this line of work, robustness requirements can be specified by signal temporal logic (STL) formulas and enforced through "classical" optimal robust controller synthesis methods [21], [13], [15]. Our *resilient ABCD* method is orthogonal to this line of work, as we *lift* the treatment of robustness to the abstract domain instead. This allows us to handle more general specification classes, namely full linear-temporal logic (LTL), and arbitrary non-linear continuous dynamics. Further, *resilient ABCD* allows to incorporate other discrete disturbances s.a. lossy channels or faulty event models [8], [25].

II. Preliminaries on Control Systems

Notation. We introduce an ordinal notation à la von Neumann. The nonnegative integers are inductively defined as $0 = \emptyset$ and $n + 1 = n \cup \{n\}$. Consequently, the first limit ordinal is the set of non-negative integers $\omega = \{0, 1, \ldots\}$ and the next two ordinals are $\omega + 1$ and $\omega + 2$. It is easy to see that these ordinals are ordered by set inclusion. For any given set S, we use the notation S^ω to represent the set of all infinite sequences that can be formed by using the elements of the set S.

Sampled-time Control System. A *sampled-time control system* is a tuple $\Sigma = (X, U, W_{\text{normal}}, f)$, which consists of a state space $X = \mathbb{R}^n$, a non-empty input space $U \subseteq \mathbb{R}^m$, a bounded disturbance set $W_{\text{normal}} \subset \mathbb{R}^n$, and a transition function $f : X \times U \times \mathbb{R}^n \rightarrow X$.

Control Specifications. Let $\mathit{Win} \subseteq X^\omega$ be a given control specification. We consider two different control specifications: safety and parity (the latter being the canonical representation of temporal logic specifications). For safety the set Win is defined by $\mathit{Safety}(W) = \{x_0 x_1 \ldots \in X^\omega \mid x_i \notin W \text{ for all } i \in \omega\}$ for a given set $W \subseteq X$ of unsafe states and requires that the systems state be always outside the set W. For parity conditions the set Win is defined by $\mathit{Parity}(\Phi) \coloneqq \{x_0 x_1 \ldots \in X^\omega \mid \limsup \Phi(x_1)\Phi(x_2)\ldots \text{ is even}\}$ for a coloring $\Phi \colon X \rightarrow \omega$ and requires that the maximum color seen by the system infinitely often be even. For notational convenience, we use Φ and W in place of $\mathit{Parity}(\Phi)$ and $\mathit{Safety}(W)$, respectively.

Both parity and safety conditions permit memoryless (or positional) control strategies, which only depend on the current state. This allows us to restrict ourselves to static state feedback controllers, introduced next.

Controller and Closed-loop. A state-feedback controller, or simply a controller, for the control system $\Sigma = (X, U, W_{\text{normal}}, f)$ is a partial function $C : X \rightarrow U$. We denote the closed loop formed by connecting C to Σ in feedback as $\Sigma \parallel C = (\mathrm{dom}(C), \mathbb{R}^n, f^C)$, where $\mathrm{dom}(C) \subseteq X$ is the domain of the controller C, the transition function $f^C : \mathrm{dom}(C) \times \mathbb{R}^n \rightarrow X$ is obtained from f by using the rule $f^C : (x, w) \mapsto f(x, C(x), w)$.

A closed-loop *trajectory* of $\Sigma \parallel C$ starting at a state $x_0 \in \mathrm{dom}(C)$, exposed to disturbances from the set $W \subseteq \mathbb{R}^n$, is an infinite sequence $\rho^W(x_0) = \{(x_0, w_0)(x_1, w_1)\ldots\}$, s.t. for all $i \in \omega$, $w_i \in W$, and $x_{i+1} = f^C(x_i, w_i)$. The trajectory $\rho^W(x_0)$ is said to satisfy a given specification Win, denoted by $\rho^W(x_0) \vDash \mathit{Win}$, if the sequence $x_0 x_1 \ldots$ satisfies Win. Otherwise $\rho^W(x_0)$ violates Win, denoted by $\rho^W(x_0) \nvDash \mathit{Win}$.

A controller C is called *sound* if for all $x \in \mathrm{dom}(C)$, and for every trajectory $\rho^{W_{\text{normal}}}(x)$ (for every possible disturbance sequence from W_{normal}), $\rho^{W_{\text{normal}}}(x) \vDash \mathit{Win}$. A sound controller C is called *maximal* if there is no other sound controller C' s.t. $\operatorname{dom}(C') \supset \operatorname{dom}(C)$. We write $\mathcal{C}^{\Sigma,\mathit{Win}}$ for the set of all sound and maximal controllers for a system Σ and a specification Win.

III. Problem Statement

We define a *risk-sensitive controller synthesis problem* using the tuple $\mathcal{P} = (\Sigma, W_{\text{high}}, \Phi)$, where $\Sigma = (X, U, W_{\text{normal}}, f)$ is a sampled-time control system that is normally exposed to disturbances from W_{normal}, the set $W_{\text{high}} \supset W_{\text{normal}}$ is a set of higher disturbance spikes that Σ is only occasionally exposed to, and Φ is a parity specification over the state space X.

For a risk-sensitive controller synthesis problem, we focus on the trajectories $\rho^{W_{\text{high}}}(x_0)$ with disturbance values from the set W_{high}, rather than $\rho^{W_{\text{normal}}}(x_0)$. Note that in practice, we interpret disturbances from the set $W_{\text{high}} \setminus W_{\text{normal}}$ as rare events. Given any trajectory $\rho^{W_{\text{high}}}(x_0) = (x_0, w_0)(x_1, w_1)\ldots$, define the operator $\mathit{NumSpikes} : \rho^{W_{\text{high}}}(x_0) \mapsto \operatorname{card}\{i \leq \omega \mid w_i \in W_{\text{high}} \setminus W_{\text{normal}}\}$, where $\operatorname{card} S$ is the cardinality of a given set S, and it is understood that W_{normal} is clear from the context. Intuitively, $\mathit{NumSpikes}(\rho^{W_{\text{high}}}(x_0))$ returns the number of times a disturbance outside W_{normal} (but in W_{high}) occurred in the trajectory $\rho^{W_{\text{high}}}(x_0)$. A closed-loop trajectory $\rho^{W_{\text{high}}}(x_0)$ is called *spike-free* if $\mathit{NumSpikes}(\rho^{W_{\text{high}}}(x_0)) = 0$.

Let $C \in \mathcal{C}^{\Sigma,\Phi}$ be a controller, and $\alpha \in \omega + 2$. We say that C is *α-resilient* from a state $x_0 \in X$ of Σ if every closed-loop trajectory $\rho^{W_{\text{high}}}(x_0)$ of $\Sigma \parallel C$ that starts in x_0, and that satisfies $\mathit{NumSpikes}(\rho^{W_{\text{high}}}(x_0)) < \alpha$ is winning. This means that a k-resilient controller with $k \in \omega$ satisfies Φ even under at most $k-1$ high disturbance spikes, an ω-resilient controller satisfies Φ even under any finite number of high disturbance spikes, and an $(\omega+1)$-resilient controller satisfies Φ even under infinitely many high disturbance spikes.

Next, we define the *resilience* of a state $x \in X$ to be

$$r_{\mathcal{P}}(x) = \sup\{\alpha \in \omega + 2 \mid \text{there exists an } \alpha\text{-resilient controller from } x\}.$$

Note that $r_{\mathcal{P}}(x) > 0$ if and only if $x \in \operatorname{dom}(C)$, because any controller is 1-resilient from x if and only if $x \in \operatorname{dom}(C)$. We call a controller $C^* \in \mathcal{C}^{\Sigma,\Phi}$ *optimally resilient*, if it is $r_{\mathcal{P}}(x)$-resilient from every state $x \in X$.

In this paper, we address the following problem: *Given a risk-sensitive controller synthesis problem $\mathcal{P}$, find a procedure that will automatically synthesize an optimally resilient controller C^*.*

Following the paradigm of abstraction-based controller synthesis (ABCS), we address this problem in three steps. We (i) construct a *risk-aware finite-state abstraction* Γ of the given *risk-sensitive controller synthesis problem* $(\Sigma, W_{\text{high}}, \Phi)$ (see Sec. IV), (ii) we synthesize an *optimally resilient controller* C for the finite state system Γ in (see Sec. V), and (iii) we refine C^* into an approximately optimally resilient controller $\widehat{C}^*$ for Σ which approaches the optimal solution if the grid size for the state and input grid goes to zero.

IV. Risk-aware Abstraction for Risk-sensitive Control Problem

A. Preliminaries

The method that we primarily build upon is the so-called Abstraction-Based Controller Synthesis techniques (ABCS). In the following, we briefly recap one of the several available ABCS techniques.

First, we introduce certain types of finite state abstract transition systems that approximates the continuous system dynamics.

Finite State Transition System. A *finite state transition system* is a tuple $\Delta = (Q, A, \delta)$ that consists of a finite set of states Q, a finite set of control actions A, and a set-valued transition map $\delta : Q \times A \rightrightarrows Q$.

Finite State Abstraction of Control Systems. A finite state transition system Δ is called a *finite state abstraction*, or simply an abstraction, of a given control system Σ if certain relation holds between the transitions of Σ and the transitions of Δ. Depending on the controller synthesis problem at hand, there are several such relations available in the literature. The one that we use in our work is the *feedback refinement relation* (FRR) [18].

An FRR is a relation $R \subseteq X \times Q$ between Σ and a finite-state transition system Δ, written as $\Sigma \preccurlyeq_R \Delta$, so that for every $(x, q) \in R$, the set of allowed control inputs (element of A) from q is a subset of the set of allowed control inputs (element of U) from x, and moreover when the same allowed control input is applied to both q and x, the image of the set of possible successors of x under R is contained in the set of successors of q.

Within this paper we assume that there is an algorithm, called $\mathit{FindAbstraction}$, which takes as input a given control system Σ and a given set of additional tuning parameters P (like the state space discretization and the control space discretization), and outputs an abstract finite state transition system Δ and an associated FRR R, s.t. $\Sigma \preccurlyeq_R \Delta$. For the actual implementation of $\mathit{FindAbstraction}(\Sigma, P)$, we refer the reader to [18].

Without going into the details of how the parameter set P influences the outcome of $\mathit{FindAbstraction}$, we would like to point out one property that we expect to hold. Let $\Sigma = (X, U, W_{\mathrm{normal}}, f)$ and $\Sigma' = (X, U, W'_{\mathrm{normal}}, f')$ be two different control systems with same state and control input spaces. Suppose $\Delta = (Q, A, \delta)$ and $\Delta' = (Q', A', \delta')$ be two transition systems computed using *FindAbstraction* using the same parameter set P i.e. $\mathit{FindAbstraction}(\Sigma, P) = \Delta$ and $\mathit{FindAbstraction}(\Sigma', P) = \Delta'$. Then there are one-to-one correspondences between Q and Q', and between A and A'. Moreover, there is an FRR R so that both $\Sigma \preccurlyeq_R \Delta$ and $\Sigma' \preccurlyeq_R \Delta'$ hold. We will abuse this property, and will use the same state space and input space for both Δ and Δ'.

B. Risk-aware Abstraction

To take into account the effect of occasional occurrences of disturbance spikes in the control system Σ, we introduce a special finite state abstract transition system, called the *bimodal transition system*, with two distinct transition relations. A bimodal transition system is a tuple $(Q, A, \delta^{\mathrm{nor}}, \delta^{\mathrm{dist}})$, where Q is a finite set of *abstract states*, A is the finite set of control actions, $\delta^{\mathrm{nor}} : Q \times A \rightrightarrows Q$ is a set-valued map representing the *normal transitions*, and $\delta^{\mathrm{dist}} : Q \times A \rightrightarrows Q$ is another set-valued map representing a set of *disturbance transitions*.

Definition 1: Let $(\Sigma, W_{\mathrm{high}}, \Phi)$ be a risk-sensitive controller synthesis problem. A bimodal transition system $\Gamma = (Q, A, \delta^{\mathrm{nor}}, \delta^{\mathrm{dist}})$ is called a *risk-aware abstraction* of Σ if there exists a relation $R \subset X \times Q$ s.t. the following conditions are satisfied:

- $\Sigma \preccurlyeq_R (Q, A, \delta^{\mathrm{nor}})$, and
- for all $(x, q) \in R$, $u \in A(q) \Rightarrow \cup_{w \in W_{\mathrm{high}}} R(f(x, u, w)) \subseteq \delta^{\mathrm{nor}}(q, u) \cup \delta^{\mathrm{dist}}(q, u)$,

where $A(q)$ is the set of control inputs allowed in the state q, i.e. $A(q) := \{u \in A \mid \delta^{\mathrm{nor}}(q, u) \neq \emptyset\}$.

Alg. 1 computes risk-aware abstraction for a given risk-sensitive controller synthesis problem and a given parameter set P. Note that, Alg. 1 can be implemented symbolically.

V. Synthesis of Optimally Resilient Controller

In the following, we present our algorithm to synthesize the optimally resilient controller for risk-aware abstractions of sampled-time control systems. Our algorithm follows the general ideas of [17], and assigns resilience to every abstract state of the risk-aware abstraction. Intuitively, the resilience of an abstract state corresponds to the maximum number of disturbance spikes that the *abstract closed-loop*—formed by connecting the synthesized controller with the abstract system in feedback—can tolerate while still satisfying its specification. We are interested to synthesize a controllers which *maximizes* the resilience of each abstract state. Before we describe our algorithm, however, let us introduce the required definitions and notation.

Algorithm 1 *FindRiskAwareAbstraction*

```
Input: $(\Sigma, W_{\mathrm{high}}, \Phi)$, some parameter set $P$ for FindAbstraction
Output: $\Gamma = (Q, A, \delta^{\mathrm{nor}}, \delta^{\mathrm{dist}})$
1: Compute $\Delta^{\mathrm{nor}} = (Q, A, \delta^{\mathrm{nor}}) \leftarrow \mathit{FindAbstraction}(\Sigma, P)$
2: Compute $\Delta^{\mathrm{high}} = (Q, A, \delta^{\mathrm{high}}) \leftarrow \mathit{FindAbstraction}\,((X, U, W_{\mathrm{high}}, f), P)$
3: For all $q \in Q$ and $u \in A$, define $\delta^{\mathrm{dist}} : (q, u) \mapsto \delta^{\mathrm{high}}(q, u) \setminus \delta^{\mathrm{nor}}(q, u)$
4: return $\Gamma = (Q, A, \delta^{\mathrm{nor}}, \delta^{\mathrm{dist}})$
```

A. Preliminaries

Let $\mathcal{P} = (\Sigma, W_{\mathrm{high}}, \Phi)$ be a given risk-sensitive controller synthesis problem, $\Gamma = (Q, A, \delta^{\mathrm{nor}}, \delta^{\mathrm{dist}})$ be a risk-aware abstraction of Σ, and R be the associated FRR between Σ and $(Q, A, \delta^{\mathrm{nor}})$. We assume that the parity specification Φ is so given and Γ is so obtained that for every abstract state $q \in Q$, every pair of associated system states $x_1, x_2 \in X$ with $(x_1, q), (x_2, q) \in R$ are assigned the same color by Φ, i.e. $\Phi(x_1) = \Phi(x_2)$. This allows us to lift the parity specification $\mathit{Parity}(\Phi)$ in a well-defined way to the abstract state space as $\mathit{Pairty}(\widehat{\Phi}) \subseteq Q^\omega$, s.t. for all $q \in Q$ and for all $x \in X$ with $(x, q) \in R$, $\widehat{\Phi} : q \mapsto \Phi(x)$. We define an *abstract risk-sensitive controller synthesis problem*, or abstract synthesis problem in short, using the tuple $\widehat{\mathcal{P}} = (\Gamma, \widehat{\Phi})$.

We export the concepts of controllers, closed-loops, closed-loop trajectories etc. from the domain of sampled-time control systems to the domain of risk-aware abstractions in the natural way. An abstract controller $\widehat{C}$ is a partial function $\widehat{C} : Q \to A$. The abstract closed-loop is obtained by connecting $\widehat{C}$ in feedback with Γ, and is defined using the tuple $\Gamma \parallel \widehat{C} = (\mathrm{dom}(\widehat{C}), \delta^{\widehat{C}})$, where $\delta^{\widehat{C}} : \mathrm{dom}(\widehat{C}) \rightrightarrows Q$ s.t. $\delta^{\widehat{C}} : q \mapsto \delta^{\mathrm{nor}}(q, \widehat{C}(q)) \cup \delta^{\mathrm{dist}}(q, \widehat{C}(q))$. The notion of trajectory, the satisfaction of specification, *NumSpikes*, spike-free trajectories, and resilience are naturally adapted for the system Γ. It should be noted, however, that *NumSpikes* and resilience are now computed by *counting the number of* δ^{dist}*-transitions* appearing in any given abstract closed-loop trace, as opposed to counting the number of disturbances appearing from the set $W_{\mathrm{high}} \setminus W_{\mathrm{normal}}$ in a given sampled-time closed-loop trace.

Like in the case of controller synthesis problem $\mathcal{P}$, we require our abstract controllers to be sound and maximal. Such a controller is called a *winning controller* w.r.t. a given abstraction specification $Win \subseteq Q^\omega$, and the respective controller domain is called the winning domain $\mathcal{W}(\widehat{\mathcal{P}})$. Let $\mathcal{C}^{\Gamma, Win}$ be the set of winning abstract controllers for the abstract synthesis problem $\widehat{\mathcal{P}} = (\Gamma, Win)$. We assume that we have access to a solver for the sound and maximal abstract controllers for the (spike-free) safety and parity specifications, which serves as a black-box method in our synthesis routine (for the actual implementation, one can use any of the available methods from the literature [24]). Such a solver takes a controller synthesis problem $\widehat{\mathcal{P}}$ with safety, parity, or a conjunction of safety and parity specification as input, and outputs the winning region $\mathcal{W}(\widehat{\mathcal{P}})$ (as well as the complement $\overline{\mathcal{W}}(\widehat{\mathcal{P}})$) together with a (uniform) abstract controller $\widehat{C}$ that is winning for every abstract state $q \in \mathcal{W}(\widehat{\mathcal{P}})$.

We define the resilience of abstract states and the optimally resilient abstract states analogously to the same for the continuous system defined in Sec. III. Recall that a k-resilient control strategy with $k \in \omega$ is winning even under at most $k-1$ disturbance spikes, an ω-resilient strategy is winning even under any finite number of disturbance spikes, and an $(\omega+1)$-resilient strategy is winning even under infinitely many disturbance spikes.

B. Computing Optimally Resilient Strategies

Following Neider, Weinert, and Zimmermann [17], we first characterize the abstract states of finite resilience. The remaining abstract states then have either resilience ω or $\omega+1$, and we show how to distinguish between them. Finally, we describe how to derive an optimally resilient abstract controller based on the computed resilient values.

1) Finite Resilience: Starting from the abstract states in $\overline{\mathcal{W}}(\widehat{\mathcal{P}})$, which have resilience 0 by definition, we use two operations to determine the abstract states of finite resilience: the disturbance update and the risk update. Intuitively, the disturbance update computes the resilience of abstract states for which a disturbance spike (i.e., a transition in δ^{dist}) leads to an abstract state whose resilience is already known. The risk update, on the other hand, determines the resilience of abstract states from which the controller can either not prevent to visit an abstract state with known resilience or it needs to move to such an abstract state in order to avoid losing.

For the remainder, let us fix a parity game $\widehat{\mathcal{P}} = (\Gamma, \widehat{\Phi})$ with risk-aware abstraction $\Gamma = (Q, A, \delta^{\mathrm{nor}}, \delta^{\mathrm{dist}})$. Following Neider, Weinert, and Zimmermann [17], we define the disturbance and risk updates as updates on partial mappings $r\colon Q \to \omega$, which are called *rankings*. Intuitively, a ranking assigns resilience to some of the abstract states. We denote the domain of r by $\mathrm{dom}(r)$ and the image of r by $\mathrm{im}(r)$.

Due to the different problem setup, we now deviate from Neider, Weinert, and Zimmermann's original algorithm in that we perform disturbance and risk updates not on the same risk-aware abstraction but on a sequence $(\Gamma_i)_{i=1,2,\ldots}$ of abstractions. We obtain Γ_{i+1} from Γ_i using an operation we call *strategy pruning*, which removes certain transitions from Γ_i that are no longer relevant for computing resilience.

Strategy pruning is inter-weaved with the disturbance and risk updates as shown in Algorithm 2. Our algorithm starts with the *initial ranking* that assign the value 0 to all $q \in \overline{\mathcal{W}}(\Gamma)$ and is otherwise undefined. Then, it computes the disturbance update on Γ_i, applies strategy pruning to obtain Γ_{i+1}, and finally computes the risk update on Γ_{i+1}. This process repeats until a fixed point is reached (i.e., $r_i = r_{i-1}$), at which point the algorithm returns the final ranking $r^* = r_i$. The ranking r^* then maps an abstract state to its resilience.

Algorithm 2 Determining finite resilience

```
Input: Risk-aware abstraction $\Gamma$
 1: Compute $\overline{\mathcal{W}}(G)$
 2: Initialize an initial ranking $r_0$ with $r_0(q) = 0$ for all
    $q \in \overline{\mathcal{W}}(G)$ and undefined otherwise
 3: $i \leftarrow 0$ and $\Gamma_0 \leftarrow \Gamma$
 4: repeat
 5:     $r' \leftarrow disturbance\_upd(r_i, \Gamma_i)$
 6:     $\Gamma_{i+1} \leftarrow strategy\_pruning(r_i, \Gamma_i)$
 7:     $r_{i+1} \leftarrow risk\_upd(r', \Gamma_{i+1})$
 8:     $i \leftarrow i + 1$
 9: until $r_i = r_{i-1}$
10: return $r^* = r_i$
```

In the remainder of this section, we describe all three operations in detail.

a) Disturbance Update: As mentioned before, the intuition behind the disturbance update, which we denote by $disturbance\_upd$, is to compute the resilience of abstract states for which a disturbance spike leads to an abstract state whose resilience is already known. It takes two inputs: a ranking r and a risk-aware abstraction $\Gamma = (Q, A, \delta^{\mathrm{nor}}, \delta^{\mathrm{dist}})$.

In the first step, the disturbance update computes for each $q \in Q$ a set $S_q \subseteq Q$ which satisfies the condition given below:

$$\begin{aligned} q' \in S_q \Leftrightarrow\ & [\text{for all } u \in A, \\ & \delta^{\text{nor}}(q,u) \neq \emptyset \text{ and } \delta^{\text{nor}}(q,u) \cap \operatorname{dom}(r) = \emptyset \\ & \text{imply } \delta^{\text{dist}}(q,u) \cap \operatorname{dom}(r) \neq \emptyset] \\ & \text{and} \\ & [\text{there exists } u \in A, \\ & \delta^{\text{nor}}(q,u) \neq \emptyset \text{ and } \delta^{\text{nor}}(q,u) \cap \operatorname{dom}(r) = \emptyset \\ & \text{and } q' \in \delta^{\text{dist}}(q,u) \text{ and } q' \in \operatorname{dom}(r)]. \end{aligned}$$

Then, it returns a new ranking r' with

$$r'(q) = \min\bigl\{\{r(q)\} \cup \{r(q')+1 \mid q' \in S_q\}\bigr\}$$

for each $q \in Q$, where $\{r(q)\} = \emptyset$ if $q \notin \operatorname{dom}(r)$, and $\min \emptyset$ is undefined.

b) Strategy Pruning: The strategy pruning step, which we denote by *strategy_pruning*, removes transitions from a risk-aware abstraction that are no longer relevant for computing resilience. It takes two inputs: a ranking r and a risk-aware abstraction $\Gamma = (Q, A, \delta^{\text{nor}}, \delta^{\text{dist}})$.

Based on the transition function δ^{nor} and δ^{dist}, strategy pruning first computes two new transition functions $\delta^{\text{nor}'}$ and $\delta^{\text{dist}'}$ where

$$\delta^{\text{nor}'}(q,u) = \begin{cases} \emptyset & \text{if } F \text{ is true; and} \\ \delta^{\text{nor}}(q,u) & \text{otherwise} \end{cases}$$

for all $q \in Q$ and $u \in A$ as well as

$$\delta^{\text{dist}'}(q,u) = \begin{cases} \emptyset & \text{if } F \text{ is true; and} \\ \delta^{\text{dist}}(q,u) & \text{otherwise} \end{cases}$$

for all $q \in Q$ and $u \in A$, and where F is true if and only if $\delta^{\text{nor}}(q,u) \cap \operatorname{dom}(r) \neq \emptyset$ or $\delta^{\text{dist}}(q,u) \cap \operatorname{dom}(r) \neq \emptyset$. Then, it returns the new risk-aware abstraction $\Gamma' = (Q, A, \delta^{\text{nor}'}, \delta^{\text{dist}'})$.

c) Risk Update: The risk update, which we denote by *risk_upd*, determines the resilience of abstract states from which the controller can either not prevent to visit an abstract state with known resilience or it needs to move to such an abstract state in order to avoid losing. Like the disturbance update, it takes two inputs: a ranking r and a risk-aware abstraction $\Gamma = (Q, A, \delta^{\text{nor}}, \delta^{\text{dist}})$.

For each $k \in \operatorname{im}(r)$, the risk update computes the set

$$\begin{aligned} B_k = \overline{\mathcal{W}}\bigl(\Gamma, & \mathit{Parity}(\Phi) \cap \\ & \mathit{Safety}(\{x \in \operatorname{dom}(r) \mid r(x) \leq k\})\bigr). \end{aligned}$$

As described above, this can be done by first solving the safety game and then the parity game on the resulting game. As a byproduct, we obtain in iteration i a controller that (i) never visits a state with resilience less than i and (ii) is winning if no disturbance spike occurs. Once the fixed point is reached, the final controller can even tolerate an arbitrary number of resilience spikes.

Then, the risk update returns the new ranking r' with

$$r'(q) = \min\{k \mid q \in B_k\},$$

for every $q \in Q$, where again $\min \emptyset$ is undefined. This concludes the computation of finite resiliences.

2) Distinguishing Between Resilience ω and $\omega+1$: It is left to distinguish between abstract states of resilience ω and $\omega+1$. Recall that abstract states of resilience $\omega+1$ are those from which the controller can satisfy the specification even if infinitely many disturbance spikes occur.

Fortunately, characterizing the abstract states of resilience $\omega+1$ is straightforward. We simply solve the classical controller synthesis problem for the abstraction $(Q, A, \delta^{\text{nor}} \cup \delta^{\text{dist}})$ (i.e., the abstraction that always allows for large disturbances), where $\delta^{\text{nor}} \cup \delta^{\text{dist}}$ denotes the argument-wise union operation of the set-valued transition functions δ^{nor} and δ^{dist}. In fact, it is then not hard to verify that the abstract states in $\operatorname{dom}(\widehat{C})$ of the resulting abstract controller $\widehat{C}$ are exactly those of resilience $\omega+1$, and the controller is $(\omega+1)$-resilient from these abstract states. This is due to the fact that $\widehat{C}$ can enforce the specification even under arbitrary many occurrences of high disturbances.

In total, we have identified the abstract states of finite resilience and resilience $\omega+1$ of a risk-aware abstraction $\Gamma = (Q, A, \delta^{\text{nor}}, \delta^{\text{dist}})$. The remaining abstract states in Q must then have resilience ω. This concludes the computation of the function r^*, which maps an abstract state to its resilience, and we can now describe how to extract an optimally resilient controller.

3) Extracting Optimally Resilient Strategies: The extraction of an optimally resilient abstract controller follows very closely the one by Neider, Weinert, and Zimmermann [17]. Intuitively, the controller extraction is the process of stitching together the controllers that were obtained in different iterations of the risk update and the $(\omega+1)$-resilience computation. The underlying idea is to switch the controller whenever a disturbance spike occurs. If finitely many disturbance spikes occur, then the optimally resilient controller will settle with an appropriate (sub-)controller that was obtained in the risk update. If infinitely many disturbance spikes occur, then the optimally resilient controller will settle with the (sub-)controller obtained in the $(\omega+1)$-resilience computation. In both cases, these (sub-)controllers are winning by construction and, hence, so is the optimally resilient one.

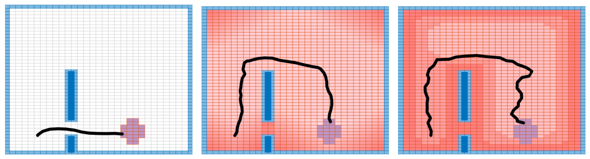

Fig. 1. **Reach-while-avoid control**: Classical ABCD using SCOTS (left); Resilient ABCD with $d = 0.5$ (middle) and $d = 2$ (right). Lower resilient states are indicated in darker shade than higher resilient states. Strategies through the wide passage are more resilient.

4) Controller refinement: Let $\mathcal{P} = (\Sigma, W_{\text{high}}, \Phi)$ be a risk-aware controller synthesis problem. Let $\widehat{C}^*$ be an optimally resilient controller synthesized for the abstract synthesis problem $\widehat{\mathcal{P}} = (\Gamma, \widehat{\Phi})$, where Γ is a risk-aware abstraction of Σ with the corresponding FRR R. Following the usual methodology of ABCD, one can define a controller for Σ as $C^* : x \mapsto \widehat{C}^*(q)$ where $q \in Q$ s.t. $(x, q) \in R$. Here, C^* is well-defined due to the properties of R. It can be shown that for every $x \in X$, C^* is an α-resilient controller for some $\alpha \leq \mathrm{r}_{\mathcal{P}}(x)$, i.e. C^* is a sub-optimal controller in terms of resilience.

VI. Experimental Results

We have implemented our *resilient ABCD* approach in the tool RESCOTS (REsilient SCOTS) which is build on top of the existing tool for ABCD called SCOTS [20] and available on gitlab.[1] We tested RESCOTS on various unicycle motion planning problems, with the unicycle dynamics taken from [18] with nominal disturbance set $W_{\text{normal}} = [-0.05, 0.05] \times [-0.05, 0.05] \times [0, 0]$. In addition, we assume that the velocity at each dimension of the unicycle will occasionally be perturbed (e.g. due to a slippery floor) by disturbances spikes from the set $W_{\text{high}} = [-d, d] \times [-d, d] \times [0, 0]$ for some $d > 0.05$.

Reach-While-Avoid. We first consider a simple *reach-while-avoid* specification for the unicycle robot as depicted in Fig. 1. In this example, the robot must pass through either a narrow or a wide passage to avoid the obstacles (blue) and reach the target (purple). We have implemented this synthesis problem as a Parity specification with three colors – color 2 assigned to target states and color 1 assigned to all other states, while color 0 is kept empty. To enforce obstacle avoidance, we manipulate the computed abstract transition system to self-loop in obstacle states on all normal and disturbance edges. Hence, whenever the robot visits an obstacle state *once*, it is forced to stay there forever and thereby visits color 1 infinitely often which violates the specification. Fig. 1 depicts the path of the robot to the target; the computed controller additionally ensures that the target is (re-) visited infinitely often, by looping inside it.

[1] `https://bitbucket.org/stanlyjs/rescot/src/master/`

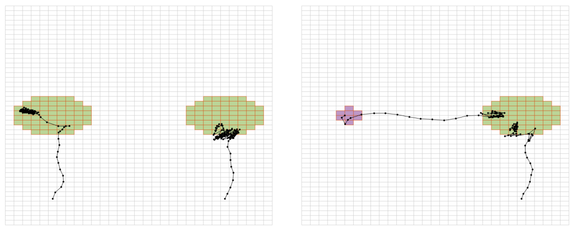

Fig. 2. **Co-Büchi vs. Büchi objectives**: Fulfilling a Co-Büchi objective for the left and a Büchi objective for the right target. ω and $\omega + 1$ resilient states are indicated in white and green, respectively.

Under normal disturbances, both passages are equally safe. Therefore, the original SCOTS algorithm returns a strategy which guides the robot along the shortest path to the target, passing the small passage (Fig. 1 (left)). In contrast, RESCOTS synthesizes a controller which enforces the shortest path through the less risky wider passage (Fig. 1 (middle and right)) while assuming occasional disturbance spikes with $d = 0.5$ (middle) and $d = 2$ (right), respectively. It can be observed that the smaller W_{high} the higher the number of resilience values: 16 for $d = 0.5$ (middle) as opposed to 3 for $d = 2$ (right). I.e., the larger the set W_{high}, the lower the number of high disturbance spikes which pushes the robot from a state in the middle of the work space into an obstacle.

Eventuality properties. To illustrate the control strategies computed by RESCOT in the presence of eventuality properties we consider a 3-color parity specification for the unicycle robot in a bounded open space with no obstacles. Here, color 0 is assigned to the left target (T_l), color 2 is assigned to the right target (T_r) and color 1 is assigned to every other state.

This specification imposes two possible strategies for the robot: (i) move to T_l (color 0) and stay inside T_l forever, or (ii) visit at least one state in T_r infinitely often. Hence, the robot can choose between a Co-Büchi specification w.r.t. T_l and a Büchi specification w.r.t. T_r.

The difference of enforced trajectories depending on the Co-Büchi and the Büchi specification is depicted in Fig. 2 (left). We see that the robot does not make an effort to get to the interior of T_r, as being pushed out by a disturbance infinitely often (and then moving back in) does not violate the specification. However, the robot makes an effort to stay inside T_l; leaving T_l infinitely often would violate the specification.

For the scenario depicted in Fig. 2 (left) both specifications are satisfyable with resilience $\omega + 1$ and the robot chooses the target closest to its starting point. If we decrease the size of the left target (Fig. 2 (right)), only the Büchi specification for T_r is satisfyable with

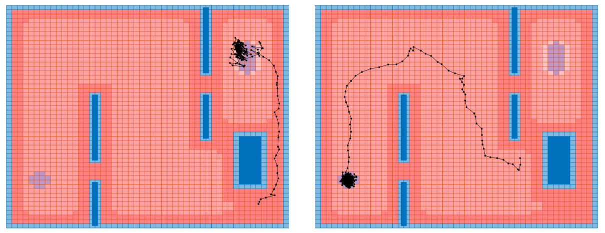
Fig. 3. Synthesis problem of Fig. 2 under additional obstacle avoidance. All states (including target states) have finite resilience. Here targets are entirely chosen based on the reaching path resilience.

resilience $\omega + 1$ and is therefore chosen by the robot from any initial state.

Eventuality properties and finite resilience. As a last experiment, we add obstacles to the scenario in Fig. 2 as shown in Fig. 3. This results in a 4-color parity game where, as before, color 0 is assigned to the left target (T_l), color 2 is assigned to the right target (T_r) and color 1 is assigned to every other state. Now we additional assign color 3 to all obstacles (blue) and again enforce obstacle avoidance by self-loops within those states.

In this case we see that all states, including the states inside the targets, get a finite resilience. This is due to the fact that a finite number of successive disturbances can force the robot to bump into the wall. Choosing to satisfy the Büchi objective (i.e., move to T_r) is therefore not more resilient then choosing to satisfy the Co-Büchi objective. In particular, the controller avoids passing the risky, dark red passage for moving to T_r and chooses to go to T_l instead, which is further away (see Fig. 3 (right)). However, if it is already inside the risky dark red region, it prefers to move to T_r, which is closer.

VII. Conclusion

We have developed a novel technique to synthesize resilient controllers for perturbed non-linear dynamical systems w.r.t. ω-regular specifications. Our approach combines two ideas to one effective algorithm: the principle of abstraction-based controller design and a recently developed algorithm for computing optimally resilient strategies in two-player games on graphs. Our experimental evaluation on various robot motion planning examples has demonstrated that our algorithm indeed produces controllers that are more resilient to disturbance spikes than classical synthesis tools, e.g., SCOTS.

In future work, we are planning to combine RESCOTS with our tool Mascot [10] to increase computational efficiency and to utilize resilient ABCD within online adaptable ABCD [1] for more realistic applications.

References

[1] Y. Bai, K. Mallik, A.-K. Schmuck, D. Zufferey, and R. Majumdar. Incremental abstraction computation for symbolic controller synthesis in a changing environment. In *CDC'19*, 2019.

[2] C. Belta, B. Yordanov, and E. A. Gol. *Formal methods for discrete-time dynamical systems*, volume 89. Springer, 2017.

[3] R. Bloem, H. Chockler, M. Ebrahimi, and O. Strichman. Synthesizing reactive systems using robustness and recovery specifications. In *FMCAD'19*, pages 147–151, 2019.

[4] R. Bloem, R. Ehlers, S. Jacobs, and R. Könighofer. How to handle assumptions in synthesis. In K. Chatterjee, R. Ehlers, and S. Jha, editors, *SYNT 2014, Vienna, Austria, July 23-24, 2014*, volume 157 of *EPTCS*, pages 34–50, 2014.

[5] E. Dallal, D. Neider, and P. Tabuada. Synthesis of safety controllers robust to unmodeled intermittent disturbances. In *CDC'16*, pages 7425–7430. IEEE, 2016.

[6] S. Dathathri, S. C. Livingston, and R. M. Murray. Enhancing tolerance to unexpected jumps in gr(1) games. In *ICCPS '17*, 2017.

[7] R. Ehlers and U. Topcu. Resilience to intermittent assumption violations in reactive synthesis. In *HSCC '14*, 2014.

[8] A. Girault and É. Rutten. Automating the addition of fault tolerance with discrete controller synthesis. *Formal Methods in System Design*, 35(2):190, 2009.

[9] S. Hagihara, A. Ueno, T. Tomita, M. Shimakawa, and N. Yonezaki. Simple synthesis of reactive systems with tolerance for unexpected environmental behavior. In *FormaliSE'16*, pages 15–21, 2016.

[10] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck. Multi-layered abstraction-based controller synthesis for continuous-time systems. In *HSCC'18*, pages 120–129. ACM, 2018.

[11] M. Khaled and M. Zamani. pfaces: an acceleration ecosystem for symbolic control. In *HSCC'19*, pages 252–257. ACM, 2019.

[12] Y. Li and J. Liu. Rocs: A robustly complete control synthesis tool for nonlinear dynamical systems. In *HSCC '18*, 2018.

[13] L. Lindemann and D. V. Dimarogonas. Robust control for signal temporal logic specifications using discrete average space robustness. *Automatica*, 101:377–387, 2019.

[14] J. Liu and N. Ozay. Finite abstractions with robustness margins for temporal logic-based control synthesis. *Nonlinear Analysis: Hybrid Systems*, 22:1 – 15, 2016.

[15] N. Mehdipour, C.-I. Vasile, and C. Belta. Arithmetic-geometric mean robustness for control from signal temporal logic specifications. In *ACC '19*, pages 1690–1695. IEEE, 2019.

[16] S. Mouelhi, A. Girard, and G. Gössler. Cosyma: a tool for controller synthesis using multi-scale abstractions. In *HSCC'13*, pages 83–88, 2013.

[17] D. Neider, A. Weinert, and M. Zimmermann. Synthesizing optimally resilient controllers. In *CSL'18*, volume 119, pages 34:1–34:17, 2018.

[18] G. Reissig, A. Weber, and M. Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE TAC*, 62(4):1781–1796, 2017.

[19] M. Rungger and P. Tabuada. A notion of robustness for cyber-physical systems. *IEEE TAC*, 61(8):2108–2123, 2016.

[20] M. Rungger and M. Zamani. Scots: A tool for the synthesis of symbolic controllers. In *HSCC'16*, pages 99–104, 2016.

[21] S. Sadraddini and C. Belta. Robust temporal logic model predictive control. In *Allerton'15*, pages 772–779. IEEE, 2015.

[22] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[23] P. Tabuada, S. Y. Caliskan, M. Rungger, and R. Majumdar. Towards robustness for cyber-physical systems. *TAC*, 59(12):3151–3163, 2014.

[24] T. van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In *TACAS'18*, 2018.

[25] X. Yu and J. Jiang. A survey of fault-tolerant controllers based on safety-related issues. *Annual Reviews in Control*, 39:46 – 57, 2015.