Discrete Event Dynamic Systems
https://doi.org/10.1007/s10626-023-00386-8

On tolerance of discrete systems with respect to transition perturbations

Rômulo Meira-Góes[1] · Eunsuk Kang[2] · Stéphane Lafortune[3] · Stavros Tripakis[4]

Received: 13 October 2022 / Accepted: 19 September 2023

Abstract
Control systems should enforce a desired property for both expected/modeled situations as well as unexpected/unmodeled environmental situations. Existing methods focus on designing controllers to enforce the desired property only when the environment behaves as expected. However, these methods lack discussion on how the system behaves when the environment is *perturbed*. In this paper, we propose an approach for analyzing discrete-state control systems with respect to their *tolerance* against environmental perturbations. We formally define this notion of tolerance and describe a general technique to compute it, for any given regular property. We also present a more efficient method to compute tolerance with respect to invariance properties. Moreover, we show that there exists an inherent trade-off between permissiveness and tolerance that we capture via Pareto optimality conditions. We also study the problem of synthesizing Pareto optimal controllers that achieve a minimum level of tolerance and permissiveness. We demonstrate our framework on examples involving surveillance protocols and robotic motion planning.

Keywords Tolerance · Discrete transition systems · Model uncertainty · Labeled transition systems

1 Introduction

In control systems, a controller is designed to enforce a desired property over the environment that it controls. Controller synthesis methods provide means to synthesize controllers that ensure a desired property expressed in formal logic (Ramadge and Wonham 1987; Pnueli and Rosner 1989; Cassandras and Lafortune 2021; Tabuada 2009; Belta et al. 2017). These controllers are usually synthesized to maximize the behavior of the closed-loop system while satisfying the property, i.e., the most *permissive* controllers. However, these methods rely

This work has been supported by the US National Science Foundation under awards CNS-1801342, CNS-1801546, CCF-1918140, and ECCS-2144416.

✉ Rômulo Meira-Góes
romulo@psu.edu

Extended author information available on the last page of the article

heavily on *assumptions* about the behavior and properties of the environment—themselves specified as part of a *formal environment model*.

In practice, building a perfectly faithful model of the environment is difficult, if not impossible, and the actual environment may behave differently from the model; we call the differences between the actual and modeled environments *perturbations*. A perturbation in the environment model may result in a violation of one or more assumptions and possibly jeopardize the correctness of the controller. Thus, in addition to being correct, an ideal controller must be capable of *tolerating* certain reasonable perturbations in the environment model, in that it is capable of guaranteeing a desired property even under those perturbations.

In this paper, we investigate the problem of analyzing the tolerance of a controller against environmental perturbations. We study this problem under controllers and environments that are modeled as *discrete transition systems*. At a highlevel, consider controller f, environment E, and desired property P such that $E|f \models P$, i.e., the controller is correct, in that it guarantees P under E. In our approach, perturbations are modeled as additional transitions to the original environment model E, resulting in a new *perturbed* model, E'. Then, the controller f is said to be *tolerant* against these perturbations if it is still capable of guaranteeing P even under this new perturbed system, i.e., $E'|f \models P$. Based on this intuition, we define the notion of *tolerance level* of a controller, denoted by Δ, as the set of *all* perturbations against which the controller is tolerant. The tolerance level is an intrinsic property of a controller that can be computed algorithmically. We show how the problem of computing tolerance can be reduced to model checking problems for discrete systems. For a general ω-regular property P, this algorithm is brute-force in nature, as it enumerates the set of all possible tolerable perturbations.

This brute-force algorithm motivates us to investigate more efficient algorithms for computing the tolerance level Δ. We do that by restricting our attention to the special case where P is an *invariance* property. Invariance properties are an important class of properties that constitute many of the applications of formal verification techniques in industry (Newcombe et al. 2015; Ball et al. 2011). Intuitively, invariance ensures that the environment remains within a set of safe states. In this case, we show that there exists a unique maximal set of perturbations for which the controller is tolerant. This result allows us to reduce the problem of computing the tolerance level Δ to a reachability analysis problem.

We also study an inherent trade-off between tolerance of controllers and their *permissiveness*. Intuitively, to achieve more tolerance, the controller needs to restrict the behavior of the controlled system, reducing permissiveness. On the other hand, to increase permissiveness, the controller enlarges the behavior of the controlled system leading to a reduction of tolerance. We formally characterize the trade-off between permissiveness and tolerance for invariance properties using Pareto optimality conditions. We show that memoryless controllers are sufficient to describe the Pareto front for invariance properties. Furthermore, we study and provide a solution to the problem of synthesizing Pareto optimal controllers that achieve a minimum level of tolerance and permissiveness.

After presenting a motivating example (Section 2) and preliminary definitions (Section 3), we define a new notion of tolerance of controllers to environmental perturbations and a general technique to automatically compute it (Section 4). In Section 5, we investigate this notion of tolerance with respect to invariance properties and devise a more efficient algorithm to compute it. We investigate the tradeoff between tolerance and permissiveness and solve a new synthesis problem in Section 6. Section 7 demonstrates our approach to examples involving surveillance protocols and robotic motion planning. Lastly, related work and conclusions are discussed in Sections 8-9, respectively. An earlier version of this paper appeared in Meira-Góes et al. (2022).

2 Motivating example

As a motivating example, we consider a surveillance scenario of two autonomous drones, ego and srv. These drones monitor the surroundings of a building as depicted in Fig. 1a. Ego desires to obtain information about the building without being captured by srv, where "captured" means that both drones are in the same location. It also assumes that srv surveils the building by following the strategy depicted in Fig. 1b, i.e., srv surveils the building by always moving in the clockwise direction.

Classical reactive synthesis techniques can synthesize a controller for ego that guarantees the satisfaction of its property (Pnueli 1977; Pnueli and Rosner 1989; Alur and La Torre 2001; Bloem et al. 2012; Grädel et al. 2002). For example, we can synthesize two controllers that guarantee that srv does not capture ego: controller 1 maintains ego most of the time in location 1 but it allows ego to visit location 2 when srv is in location 4, and controller 2 ensures that ego trails two steps behind srv. These controllers, however, may no longer guarantee the property if the model of the system changes.

Suppose that srv does not conform with the strategy assumed by ego, e.g., srv decides to go counter-clockwise to monitor the building. Then, we must validate these controllers against these perturbations with an extra verification step. For example, if srv moves counter-clockwise, controller 1 still guarantees that ego is not captured but controller 2 no longer does. Another option is to synthesize a new controller based on the model of the system augmented with the possible "known" perturbations (Topcu et al. 2012).

In comparison, we pose the following question: *For which model perturbations will a controller ensure the given property?* Our notion of tolerance level, Δ, answers this question. Δ is useful for system designers in a number of ways:

(1) Understanding controller tolerance Δ exactly captures the set of all perturbations under which the system can (and cannot) guarantee a given property. For example, our notion of tolerance explicitly states that controller 1 ensures ego's property even when srv moves counter-clockwise. On the other hand, the tolerance of controller 2 states that ego will collide with srv when srv moves counter-clockwise. The designer can analyze these controllers before their deployment. If Δ does not include certain perturbations of interest, e.g., srv moving counter-clockwise, the designer may revise the controller to increase its tolerance to a desired level.

(2) Comparing tolerance and permissiveness of different controllers Our notion of tolerance also allows comparison between controllers (f_1 and f_2) with respect to their tolerance levels, e.g., $\Delta_1 \subset \Delta_2$? For example, this comparison allows us to affirm that controller 1 is

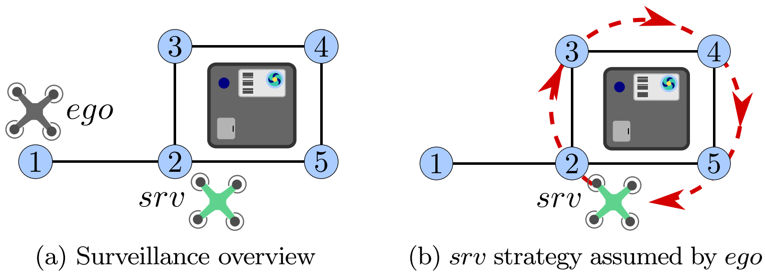


Fig. 1 Motivating example of a surveillance scenario

more tolerant than controller 2. This tolerance comparison can be enhanced with the notion of permissiveness. For instance, although controller 1 is more tolerant than controller 2, this additional tolerance comes at the cost of permissiveness. Controller 1 only allows ego to visit location 2 while controller 2 allows ego to visit locations 2-5. The designer can analyze this trade-off between permissiveness and tolerance using our Pareto optimality conditions. In this manner, the designer may decide which controller to deploy based on both the tolerance and permissiveness of controllers by comparing different design choices.

(3) Synthesizing tolerant and permissive controllers Since we develop mechanisms to compare different controllers based on tolerance and permissiveness, our framework also allows the designer to automatically synthesize controllers that achieve desired levels of tolerance and permissiveness by construction. For example, a possible design flow using our synthesis result is as follows. The designer starts with the initial controller design of controller 2. After analyzing the tolerance of controller 2, the designer realizes that this controller is not tolerant when srv moves counter-clockwise. Next, the designer uses our synthesis tool to specify that ego must be tolerant when srv moves counter-clockwise and it should visit at least one of the states in 2-5. Our synthesis tool automatically computes controller 1 which satisfies by construction the above requirements.

3 Discrete-state systems

This section describes the underlying formalism used to model the environment, controlled systems, and the properties enforced by them.

3.1 Labeled transition systems

In this work, we use labeled transition systems to model the behavior of the environment.

Definition 1 A *labeled transition system* (LTS) T is a tuple $\langle Q, Act, R, I\rangle$, where Q is a finite set of states, Act is a finite set of actions, $R \subseteq Q \times Act \times Q$ is the transition relation of T, and $I \subseteq Q$ is a nonempty set of initial states.

Let $Post_T(q, a)$ denote the set of immediate successor states from state $q \in Q$ and action $a \in Act$, i.e., $Post_T(q, a) := \{q' \in Q \mid (q, a, q') \in R\}$. A *run* of T starts at an initial state in I and is followed by a finite or infinite alternating sequence of actions and states complying with the transitions in R, e.g., $x_0a_0x_1a_1 \ldots x_n$ such that $x_{i+1} \in Post_T(x_i, a_i)$ for all $i < n$ and $x_0 \in I$. The set of all runs in T is denoted by $Runs(T)$. A *path* of T is the sequence of states in a run of T, e.g., for $x_0a_0x_1 \in Runs(T)$, then x_0x_1 is a path of T. The set of all, finite or infinite, paths in T is denoted by $Paths(T)$. We also denote the set of finite paths as $Paths_{fin}(T)$.

Example 1 We model the motivating example in Section 2 using LTS. The states represent the discrete locations of ego, $\{1, 2, 3, 4, 5\}$ and srv, $\{2, 3, 4, 5\}$. The possible actions of the system consist of ego selecting its desired next location, i.e., $Act = \{m1, \ldots, m5\}$ where mi means that ego moves to location i. The transition relation is defined by a few update rules and assumptions. The two drones move synchronously to their next location. Next, both drones can only move to locations that are connected by an edge in Fig. 1a. Lastly, we assume that srv surveils the building using the strategy defined in Fig. 1b, e.g., srv moves to location 2 when 5 is its current location, and so forth. The system is initialized in state $(1, 5)$, i.e., ego in location 1 and srv in location 5. Figure 2a partially depicts the LTS T defined by this example.

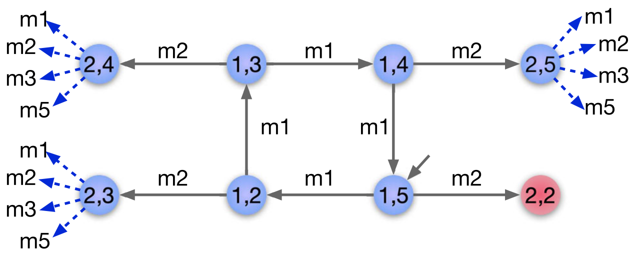
Remark 1 Our definition of LTS assumes that all elements of the set of actions Act are "controllable" actions, that can be acted upon by a controller (defined below). However, the nondeterministic transition relation of T can be used to model uncontrollable actions of the environment. After an action a is selected by the controller at state q, the environment decides which state the system will be in; this is similar to two-player games (Grädel et al. 2002). This can be modeled by adding several transitions from q, all labeled with the same action a.

Given a finite set A, the usual notations $|A|$, A^*, A^+, and A^ω denote the cardinality of A, the set of all finite sequences, the set of all non-empty finite sequences, and the set of all infinite sequences of elements in A, respectively. For convenience, we write $x_{0\dots n}$ for any finite sequence of states $x_0 \dots x_n$.

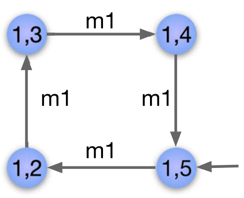
3.2 Control strategy

Given an LTS T, a control strategy, or simply *controller*, for T is a function that maps a finite sequence of states to a set of actions, i.e., $f : Q^+ \to 2^{Act}$. A *controlled run* of T is a run of T, $x_0a_0 \cdots \in Runs(T)$, such that $a_i \in f(x_{0\dots i})$ for any $i \geq 0$, i.e., the controller constrains which actions are executed by T. The set of all controlled runs, denoted by $Runs(T|f)$, defines the closed-loop system of f controlling T. For convenience, this closed-loop system is denoted by $T|f$. The set of all, finite and infinite, controlled paths is denoted by $Paths(T|f)$. We also denote the set of finite controlled paths as $Paths_{fin}(T|f)$. A controller has *finite memory* if its decisions depend only on a finite number of states. It is *memoryless* if its decisions depend only on the current state, $f\, Q \to 2^{Act}$. When f has finite memory, $T|f$ can be represented by an LTS; see the Appendix for details.

Example 2 Back to our motivating example, we give an example of a simple memoryless controller that is set to maintain ego in location 1. Formally, the controller is defined as $f(1,i) = f(2,i) = \{m1\}$ for $i \in \{2,\dots,5\}$, $f(3,4) = f(5,4) = \{m2\}$, $f(3,5) = f(4,5) = \{m3\}$, $f(3,2) = f(4,2) = f(5,2) = \{m4\}$, $f(4,3) = f(5,3) = \{m5\}$, otherwise $f(q) = \emptyset$. Figure 2b shows the reachable states of the LTS representation of $T|f$ when $I = \{(1,5)\}$.



(a) Partial LTS of the surveillance example. States are (ego loc.,srv loc.) and edge labels are the ego actions. Missing transitions are in blue.



(b) Representation of $T|f$

Fig. 2 LTSs of the surveillance scenario

3.3 Property

In this work, we consider the class of linear-time (LT) properties over the set of states Q of a given LTS T (Baier and Katoen 2008). In words, an LT property is a set of infinite or finite sequences of states that represents an "admissible/desired" set of paths of T. We recall some definitions of LT properties (Baier and Katoen 2008).

Definition 2 A *linear-time* (LT) property over the set of states Q is a subset $P \subseteq Q^\omega \cup Q^*$.

Example 3 Using our surveillance example, we want ego to not be captured by srv. We can formally define an LT property to capture this behavior. Based on the LTS model given in Example 1, we assume that srv captures ego when they share the same location, e.g., state $(2, 2)$. Let $Q_{uns} = \{(2, 2), (3, 3), (4, 4), (5, 5)\}$ be the set of unsafe states. The property $P_{srv} = (Q \setminus Q_{uns})^\omega \cup (Q \setminus Q_{uns})^*$ defines the property that ego is not captured by srv.

Based on an LT property, we can verify if an LTS satisfies this property. Intuitively, the LTS satisfies a property if its paths are all contained in the desired property. Formally, we have:

Definition 3 An LTS T satisfies property P, denoted by $T \models P$, if and only if $Paths(T) \subseteq P$.

Similarly, a controlled system $T|f$ satisfies property P if $Paths(T|f) \subseteq P$. The problem of finding a controller f such that $T|f \models P$ has been widely investigated (Ramadge and Wonham 1987; Pnueli and Rosner 1989; Ehlers et al. 2017).

Example 4 Using our surveillance example, one can verify that the controlled system $T|f$ in Fig. 2b satisfies property P_{srv}, i.e., $T|f \models P_{srv}$.

4 Tolerance against perturbations

4.1 Perturbations

Model-based control theory methods are grounded on a model of the environment under control. This model is always an approximation of the true system. For this reason, we must take into account possible mismatches between the model of the environment and the true environment when designing a controller. In the case of LTS, we model these possible mismatches, called *perturbations*, as additional transitions.

Adding transitions to the original environment introduces new behaviors to the environment that can potentially generate unsafe behavior in the controlled system. For this reason, we only consider adding new transitions to the environment. For example, transition $\big((1, 2), m1, (1, 5)\big)$ represents a perturbation to the strategy of srv, as depicted in Fig. 1b: srv goes back to position 5 instead of going to position 3. This type of perturbation can potentially catch ego off-guard and lead to its capture since ego expects srv to move to position 3. A second type of perturbation is transition $\big((1, 2), m1, (2, 3)\big)$ where ego gets pushed to location 2, e.g., by a wind gust, even though it has selected to stay in location 1.

Formally, *a perturbation* is a set of transitions $d \subseteq (Q \times Act \times Q)$. For simplicity, we define perturbations without removing the transition relation R to not overload our definitions with the removal of R. All of our results hold when perturbations are defined as $d \subseteq (Q \times Act \times Q) \setminus R$. Given a perturbation set, we can define the *perturbed system* by augmenting the transition relation of the LTS with the perturbation set.

Definition 4 Let an LTS $T = \langle Q, Act, R, I\rangle$ and a perturbation $d \subseteq Q \times Act \times Q$ be given. We define the *perturbed system* T_d as $T_d := \langle Q, Act, R \cup d, I\rangle$.

A controller f that guarantees property P for system T, $T|f \models P$, might violate this property for the perturbed system T_d. Thus, one needs to check if f continues to satisfy P for T_d, i.e., if $T_d|f \models P$ or not.

Definition 5 Controller f is a *tolerant controller* with respect to an LTS T, a perturbation d, and a property P if $T_d|f \models P$. Perturbation d is a *tolerable perturbation* with respect to T, f, and P if f is a tolerant controller with respect to T, d, and P.

Remark 2 Our definition of perturbed systems only allows adding transitions to the original environment. We do not consider perturbations that remove transitions from the original environment. However, the results in this section can be generalized to consider removing transitions with minor modifications to our definitions. One of the reasons to only allow adding transitions is related to safety-critical systems. When dealing with safety properties as we will in Section 5, it is sufficient to only consider adding new transitions to the environment. If a controlled system is safe, then deleting transitions from the environment preserves safety.

4.2 Comparing perturbations

Given perturbations d_1 and d_2 such that $d_1 \subseteq d_2$, d_2 perturbs LTS T more than d_1 since $Runs(T_{d_1}) \subseteq Runs(T_{d_2})$. Our definition of tolerable perturbations takes into account not only the perturbed system, but a controller f and its controlled behavior, e.g., $T_{d_1}|f$. By including the controller to close the loop, two incomparable perturbations can generate a comparable set of runs, i.e., it might be that $Runs(T_{d_1}|f) \subseteq Runs(T_{d_2}|f)$ even when $d_1 \nsubseteq d_2$ and $d_2 \nsubseteq d_1$. In this scenario, d_2 perturbs the controlled system more than d_1 since d_2 has more influence on the controlled behavior. Moreover, whenever $d_1 \subseteq d_2$, it follows that $Runs(T_{d_1}|f) \subseteq Runs(T_{d_2}|f)$ for any controller f. Based on this discussion, we propose a novel definition that captures formally the notion of a perturbation being more "powerful" than another one.

Definition 6 Let an LTS T, a controller f, and perturbations d_1 and d_2 be given. We say d_1 is *at least as powerful* as d_2 with respect to f, denoted by $d_2 \preceq_f d_1$, if

(i) $Runs(T_{d_2}|f) \subset Runs(T_{d_1}|f)$; or
(ii) $Runs(T_{d_2}|f) = Runs(T_{d_1}|f) \ \wedge \ d_2 \subseteq d_1$.

Whenever the controller f is clear from the context, we write $\preceq$ instead of $\preceq_f$.

Intuitively, perturbation d_1 is at least as powerful as perturbation d_2 with respect to controller f, if the controlled perturbed system $T_{d_1}|f$ can generate strictly more runs than $T_{d_2}|f$, or the two controlled systems generate exactly the same set of runs and $d_2 \subseteq d_1$. The ordering $\preceq$ forms a partial order over the set of perturbations of T. To provide more intuition on $\preceq$, we give the following example.

Example 5 Consider the LTS T shown in Fig. 3a and the property defined by all sequence of states that do not reach state 3, e.g., the sequence 143 violates this property. We define the memoryless controller f as $f(q) = \{b\}$ if $q \neq 3$, and $f(3) = \emptyset$. It follows that f satisfies the stated property, i.e., $T|f \models P$.

Consider the tolerable perturbations $d_1 = \{(1, b, 2)\}$, $d_2 = \{(1, b, 4)\}$, $d_3 = \{(2, b, 3)\}$, and $d_4 = \{(4, b, 3)\}$. Perturbations d_1 and d_2 are at least as powerful as d_3 and d_4, i.e.,

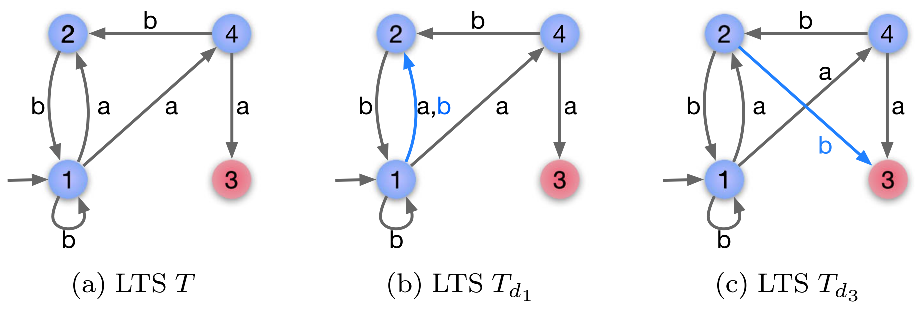


Fig. 3 Tolerable perturbations

$d_3 \preceq d_1, d_4 \preceq d_1, d_3 \preceq d_2$, and $d_4 \preceq d_2$. On the other hand, d_1 and d_2 are incomparable with respect to $\preceq$ as their perturbed controlled systems generate incomparable runs. Perturbations d_3 and d_4 are also incomparable even though $Runs(T_{d_3}|f) = Runs(T_{d_4}|f)$. In this case, condition (ii) in Definition 6 is violated as $d_3 \nsubseteq d_4$ and $d_4 \nsubseteq d_3$.

4.3 Tolerance

Intuitively, we search for all possible tolerable perturbations d with respect to LTS T, controller f, and property P.

Definition 7 Let an LTS T, a property P, and a controller f such that $T|f \models P$ be given. The tolerance of f with respect to P and T, denoted by $\Delta(T, f, P)$, is a set of perturbations $\Delta(T, f, P) \subseteq 2^{Q\times Act\times Q}$. $\Delta(T, f, P)$ is defined to be the set of perturbations satisfying the following conditions:

1. $\forall d \in \Delta(T, f, P).\ T_d|f \models P$ *[d is tolerable]*;
2. $\forall d \subseteq Q \times Act \times Q.\ T_d|f \models P \Rightarrow \exists d' \in \Delta(T, f, P).\ d \preceq d'$ *[d is represented]*;
3. $\forall d, d' \in \Delta(T, f, P).\ d \neq d' \Rightarrow d \npreceq d'$ *[unique representation]*.

Conditions 2 and 3 in Definition 7 enforce that only maximal tolerable perturbations with respect to $\preceq$ are in Δ. Formally, the set Δ defines an antichain, with respect to $\preceq$, of maximal tolerable perturbations. Intuitively, the set Δ defines an upper bound on the possible perturbations from T that controller f tolerates. Note that Δ is always nonempty since we assume that $T|f \models P$, i.e., R is always tolerable. In order for the definition of Δ to be valid, we must show that there is a unique set of perturbations that satisfies the conditions of Definition 7. This is ensured by the following result:

Lemma 1 *Given an LTS T, a controller f, and a property P, there is a unique $\Delta(T, f, P)$ that satisfies the conditions in Definition 7.*

Proof By contradiction. Assume that there exist $\Delta_1, \Delta_2 \subseteq 2^{Q\times Act\times Q}$ such that they satisfy conditions 1, 2, and 3 in Definition 7 and $\Delta_1 \neq \Delta_2$. Without loss of generality, we assume that $\exists d_1 \in \Delta_1 \setminus \Delta_2$. Since $d_1 \in \Delta_1$, we have that $T_{d_1}|f \models P$ as Δ_1 satisfies 1. As Δ_2 satisfies 2 and $d_1 \notin \Delta_2$, we have that $\exists d_2 \in \Delta_2$ such that $T_{d_2}|f \models P$ and $Runs(T_{d_1}|f) \subset Runs(T_{d_2}|f)$ or $d_1 \subseteq d_2$ ($d_1 \preceq d_2$). Since $d_1 \in \Delta_1 \setminus \Delta_2$, it follows that $Runs(T_{d_1}|f) \subset Runs(T_{d_2}|f)$ or $d_1 \subset d_2$. Back to Δ_1, condition 3 implies that $d_2 \notin \Delta_1$ since $d_1 \in \Delta_1$ and $Runs(T_{d_1}|f) \subseteq Runs(T_{d_2}|f)$. Furthermore, it does not exist $d \in \Delta_1$ such that $Runs(T_{d_2}|f) \subset Runs(T_d|f)$

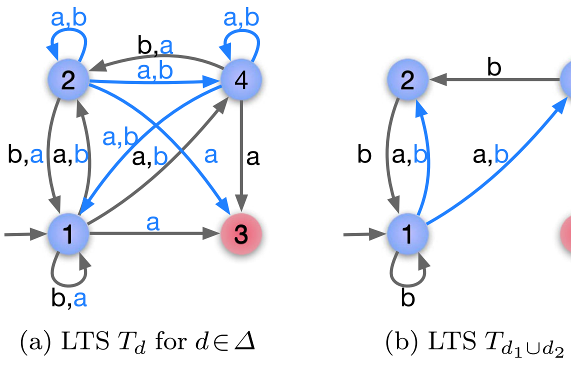


Fig. 4 LTS in Examples 6 and 7

or $d_2 \subseteq d$, because $d_1 \in \Delta_1$ and $Runs(T_{d_1}|f) \subseteq Runs(T_{d_2}|f) \subseteq Runs(T_d|f)$, and Δ_1 satisfies condition 3. Consequently, the perturbation d_2 is a witness of the Δ_1 violating condition 2, which contradicts our assumption that Δ_1 satisfies conditions 1, 2, and 3. □

Example 6 Consider the same setup as in Example 5. The four perturbations in Example 5 are tolerable. Therefore, they must be represented in Δ as stated in condition (2) in the definition of Δ. At this moment, we simply provide Δ for this example and in Section 5 we provide the formal results on efficiently obtaining this Δ. The set Δ in this example is given by $\Delta = \{Q \times Act \times Q \setminus \{(1, b, 3), (2, b, 3), (4, b, 3)\}\}$. Intuitively, Δ is defined by a single perturbation set that contains all possible transitions except the ones from states 1, 2, 4 to state 3 with action b. Adding any of these missing transitions make the perturbation set in Δ to be intolerable. The perturbed system defined by this perturbation is depicted in Fig. 4a where we highlight the new transitions in blue[1]. Any other tolerable perturbation is represented in Δ. For example, perturbations $d_1, d_2 \subseteq d = Q \times Act \times Q \setminus \{(1, b, 3), (2, b, 3), (4, b, 3)\}$ which implies that $d_1, d_2 \preceq_f d$. And although d_3 and d_4 are not subsets of d, it also follows that $d_3 \preceq d$ and $d_4 \preceq d$ since $Runs(T_{d_3}|f) = Runs(T_{d_4}|f) \subset Runs(T_d|f)$.

4.4 Computing tolerance for general properties

The tolerance of controller f is defined by the set of maximal tolerable perturbations with respect to property P. The first problem we investigate is to compute the set Δ given T, f, and P.

Problem 1 Given an LTS T, a property P, and a controller f, compute $\Delta(T, f, P)$.

Assuming that P is either a regular language or an ω-regular language, and that f has finite memory, Problem 1 is decidable; and Algorithm 1 provides a *brute force* solution to this problem. Intuitively, Algorithm 1 is broken into (i) finding the set of all tolerable perturbations (lines 2-4) and (ii) identifying the maximal ones within this set (lines 5-6). The verification tasks in lines 3 and 5 can be solved using standard model checking techniques (Baier and Katoen 2008).

Although Algorithm 1 computes the tolerance of f, this brute force method will not scale for large LTS. For this reason, we investigate more efficient ways to compute the set

[1] For simplicity, we do not show the transitions starting in state 3.

$\Delta(T, f, P)$. In the next section, we provide a more efficient algorithm for the case where P is an invariance property.

Algorithm 1 Compute-tolerance.

```
Input: $T$, $f$, and $P$
Output: $\Delta$
1: $\Delta := \emptyset$
2: for all perturbations $d \subseteq Q \times Act \times Q$ do
3:     if $T_d | f \models P$ then
4:         $\Delta := \Delta \cup \{d\}$
5: while $\exists d_1, d_2 \in \Delta$ s.t. $d_1 \preceq_f d_2$ do
6:     $\Delta := \Delta \setminus \{d_1\}$
   return $\Delta$
```

5 Tolerance with respect to invariance properties

Invariance properties are an important class of properties in industrial practice (Newcombe et al. 2015; Ball et al. 2011). An *invariance property* P for an LTS T can be represented by a subset of safe states $Q_{inv} \subseteq Q$ (Baier and Katoen 2008). Formally, a property P is an invariance property if there exists a set of safe states Q_{inv} such that $P = Q_{inv}^* \cup Q_{inv}^\omega$. For instance, $Q_{inv} = \{1, 2, 4\}$ in Example 5. An LTS satisfies an invariance property if and only if the LTS only reaches states in Q_{inv} (Baier and Katoen 2008). For convenience, we assume that the safe set of states always contains the set of initial states.

5.1 Supremum tolerable perturbation

Usually when dealing with invariance properties, one can show the existence of a single supremum element that satisfies the desired investigated property. In our scenario, we want to show that the tolerance of f with respect to an invariance property is represented by a unique tolerable perturbation, i.e., $|\Delta(T, f, P)| = 1$. Although Δ in Example 6 has a single element, the following counterexample illustrates that in general $|\Delta(T, f, P)| \geq 1$.

Example 7 Consider the setup of Example 5 with the LTS defined in Fig. 3a and $Q_{inv} = \{1, 2, 4\}$, but under control of the following controller: $f(1214) = \{a\}$ and $f(x_{0\ldots n}) = \{b\}$ for any $x_{0\ldots n} \in Q^+$ other than 1214. Perturbations $d_1 = \{(1, b, 2)\}$ and $d_2 = \{(1, b, 4)\}$ remain tolerable with respect to this new controller. And although these perturbations are tolerable, their union is not tolerable since path 1214 becomes feasible in $T_{d_1 \cup d_2}$ as seen in Fig. 4b. The size of $\Delta(T, f, P)$ must be at least two since we cannot combine d_1 and d_2 as a single tolerable perturbation that generates the behavior of $T_{d_1}|f$ and $T_{d_2}|f$.

5.1.1 Invariant controllers

The counterexample in Example 7 sheds light on the problem of the controller f selecting "bad" control decisions for paths outside of $Paths_{fin}(T|f)$. This problem can be easily fixed for invariance properties by introducing the notion of *invariant control actions* and *invariant controllers*.

Definition 8 Let an LTS T and an invariance property P with set of safe states Q_{inv} be given. The set of *invariant control actions* is defined as $A_{inv}(q) := \{a \in Act \mid Post_T(q, a) \subseteq Q_{inv}\}$ if $q \in Q_{inv}$ and $A_{inv}(q) := \emptyset$ if $q \notin Q_{inv}$. Moreover, we say that f is an *invariant controller* with respect to T and P if $f(x_{0\dots n}) \subseteq A_{inv}(x_n)$ for any sequence $x_{0\dots n} \in Q^+$.

Informally, invariant control actions characterize the "good" actions with respect to LTS T and invariance property P. Therefore, all invariant controllers satisfy invariance property P as stated in Lemma 2.

Lemma 2 *Any invariant controller f with respect to T and an invariance property P satisfies P, i.e., $T|f \models P$.*

Proof It directly follows from the definition of invariant controllers (Definition 8). □

Tolerance of invariant controllers

Under the assumption of invariant controllers, the tolerance of a given controller f is completely defined by a *unique tolerable perturbation*, i.e., $|\Delta(T, f, T)| = 1$ for any invariant controller f. We formalize this statement in the following theorem.

Theorem 1 *Let an LTS T, an invariance property P with set of safe states Q_{inv}, and an invariant controller f be given. It follows that $\Delta(T, f, P) = \{\lceil f \rceil\}$, where $\lceil f \rceil$ is defined as:*

$$\lceil f \rceil := (Q \times Act \times Q) \setminus \{(q, a, q') \in Q_{inv} \times F(q) \times (Q \setminus Q_{inv})\}$$

where $F(q) := \{a \in Act \mid \exists x_{0\dots n} \in Paths_{fin}(T_\Omega|f).\ a \in f(x_{0\dots n}) \wedge q = x_n\}$ and where $\Omega := Q_{inv} \times Act \times Q_{inv}$.

Proof We first show by contradiction that $|\Delta(T, f, P)| = 1$. For simplicity, we write Δ instead of $\Delta(T, f, P)$. Since $\emptyset$ is always a tolerable perturbation, it follows that $|\Delta| \geq 1$. Assume that $|\Delta| > 1$ and let $d_1, d_2 \in \Delta$. In the definition of Δ, condition (3) states that $d_1 \npreceq d_2$ and $d_2 \npreceq d_1$. We define $d_i^{inv} := \{(q, a, q') \in d_i \mid q, q' \in Q_{inv} \wedge a \in A_{inv}(q)\}$ for $i \in \{1, 2\}$. By construction, the controlled system $T_{d_i^{inv}}|f$ generates the same runs as $T_{d_i}|f$ for $i \in \{1, 2\}$ otherwise d_i is not tolerable. As $d_1, d_2 \in \Delta$, it must be that d_1^{inv} and d_2^{inv} are incomparable, otherwise $d_1 \preceq d_2$ or $d_2 \preceq d_1$. Because d_1^{inv} and d_2^{inv} only define transitions within Q_{inv} and f is invariant, we have that $d_1^{inv} \cup d_2^{inv}$ is a tolerable perturbation, i.e., $T_{d_1^{inv} \cup d_2^{inv}}|f \models P$. The perturbation $d_1^{inv} \cup d_2^{inv}$ must be represented in Δ as stated by condition (2) in Definition 7. Since d_1^{inv} and d_2^{inv} are incomparable, the representation of $d_1^{inv} \cup d_2^{inv}$ must be different than d_1 and d_2. Thus, there exist $d_3 \in \Delta$ different than d_1 and d_2 such that $d_1^{inv} \cup d_2^{inv} \preceq d_3$. Since the condition $Runs(T_{d_1^{inv}}|f) = Runs(T_{d_1}|f) \subset Runs(T_{d_1^{inv} \cup d_2^{inv}}|f)$, it follows that $d_1 \preceq d_3$, which violates condition (3) in the definition of Δ. That is, we have two perturbation sets in Δ that are comparable via $\preceq_f$. We reached a contradiction.

Next, we show by contradiction that $\lceil f \rceil \in \Delta$. Assume that perturbation $d \neq \lceil f \rceil$ satisfies $T_d|f \models P$ and $d \in \Delta$. By construction of $\lceil f \rceil$, the runs generated by $T_{\lceil f \rceil}|f$ are the same as the ones generated by $T_\Omega|f$. Therefore, the perturbation $\lceil f \rceil$ is tolerable. We have shown previously that $|\Delta| = 1$, which ensures that $\lceil f \rceil \preceq d$ since $d \in \Delta$. Therefore, it must be that $Runs(T_{\lceil f \rceil}|f) \subset Runs(T_d|f)$ or $Runs(T_{\lceil f \rceil}|f) = Runs(T_d|f)$ and $\lceil f \rceil \subset d$. If $Runs(T_{\lceil f \rceil}|f) \subset Runs(T_d|f)$, then d is not a tolerable perturbation since $Runs(T_{\lceil f \rceil}|f) = Runs(T_\Omega|f)$. If $Runs(T_{\lceil f \rceil}|f) = Runs(T_d|f)$ and $\lceil f \rceil \subset d$, then there exists a transition in d that is not in $\lceil f \rceil$ and this transition is not active in any run. However, by the definition of $\lceil f \rceil$, any transition in d that is not in $\lceil f \rceil$ implies that $Runs(T_{\lceil f \rceil}|f) \subset Runs(T_d|f)$ and d not being a tolerable perturbation. It follows that d is not a tolerable perturbation, which contradicts our assumption that $d \in \Delta$. □

Theorem 1 states that the tolerance of f has a *single perturbation*, i.e., there exists a supremal element within the set of tolerable perturbations with respect to $\preceq_f$. This perturbation is defined by removing transitions that are not tolerable from the set of all possible transitions. For this reason, the removed transitions are from states in Q_{inv} to states outside of Q_{inv}.

Discussing $\lceil f \rceil$ in more detail, the function $F(q)$ restricts attention to paths in $T_\Omega|f$. Recall that relation $\preceq_f$ prioritizes the behavior generated by a perturbed controlled system, i.e., $T_d|f$. The tolerable perturbation Ω is selected since it can make every state in the the safe set reachable, i.e., more behavior can be generated. Next, we investigate which actions the controller uses in the safe states reached in $T_\Omega|f$. Intuitively, if the controller uses action a in a reachable safe state q, then the transitions in $\{q\} \times \{a\} \times Q \setminus Q_{inv}$ are not tolerable and thus they must be removed from $\lceil f \rceil$.

Example 8 We return to Example 6 to discuss Theorem 1. The LTS T is depicted in Fig. 3a, the invariance property P is defined by the set $Q_{inv} = \{1, 2, 4\}$, and invariant controller f is defined as $f(q) = \{b\}$ if $q \in Q_{inv}$ and $f(3) = \emptyset$. It follows that $F(q)$ is equal to $f(q)$ for any $q \in Q$. Intuitively, the function F defines which actions the controller uses in each safe state, e.g., action b is used in state 1. Since the controller uses action b in state 1, the system is not tolerant if it is perturbed by transition $(1, b, 3)$. Similarly, action b is also used in states 2 and 4 which results in $\lceil f \rceil = Q \times Act \times Q \setminus \{(1, b, 3), (2, b, 3), (4, b, 3)\}$. Figure 4a depicts the perturbed system $T_{\lceil f \rceil}$.

5.2 Computing tolerance for invariance properties

Problem 1 investigates the computation of the set Δ for a general property P. We specialize Problem 1 to invariance properties as to use the results of Theorem 1.

Problem 2 Given an LTS T, an invariance property P, and an invariant controller f, compute $\Delta(T, f, P)$.

According to Theorem 1, for invariance property P and invariant controller f, $\Delta(T, f, P) = \{\lceil f \rceil\}$. Therefore, it suffices to compute $\lceil f \rceil$. For simplicity, we describe the computation of $\lceil f \rceil$ for memoryless controllers, but our algorithm can be extended to controllers with memory using the LTS definition of $T|f$ in the Appendix. Intuitively, Algorithm 2 performs a reachability analysis of the perturbed system $T_\Omega|f$ as to compute the function $F(q)$. Algorithm 2 is linear in the number of states and transitions of the LTS T_Ω (Baier and Katoen 2008).

Remark 3 The verification of regular safety properties can usually be transformed into a problem of verification of an invariance property. This invariance property is obtained by first composing the environment with the safety property (Baier and Katoen 2008). In this composed system, an invariance property is simply defined by a set of safe states. Unfortunately, computing robustness for safety properties does not directly reduce to computing robustness for invariance properties. The states in the composed system are tuples $(Envstate, Pstate)$. Thus, the transformation procedure introduces memory to the environment to differentiate when the safety property is violated or not. This memory is not part of the environment and prevents the direct use of Algorithm 2. We leave investigating the computation of robustness for regular safety properties to future work.

Algorithm 2 compute-invariance-tolerance.

```
Input: $T$, $f$, and $Q_{inv}$
Output: $\lceil f \rceil$
1: $\lceil f \rceil := Q \times Act \times Q$; $U := R := \{q_0\}$
2: while $U \neq \emptyset$ do
3:     pick some $q \in U$; $U := U \setminus \{q\}$
4:     for all $a \in f(q)$ do
5:         $\lceil f \rceil := \lceil f \rceil \setminus \{\{q\} \times \{a\} \times (Q \setminus Q_{inv})\}$
6:         for all $q' \in Post_{T_\Omega}(q, a)$ do
7:             if $q' \notin R$ then
8:                 $R := R \cup \{q'\}$; $U := U \cup \{q'\}$
   return $\lceil f \rceil$
```

5.3 The least and most tolerant invariant controllers

There is an inherent trade-off between tolerance and the restriction controller f imposes on LTS T. Controllers that are more *permissive* (Cassandras and Lafortune 2021), i.e., that allow more behaviors on T, are necessarily less tolerant and vice-versa. The two extremes of this trade-off are the least and the most tolerant invariant controllers. These are controllers f_1 and f_2 that satisfy $\lceil f_1 \rceil \subseteq \lceil f \rceil \subseteq \lceil f_2 \rceil$ for any other invariant controller f.

Definition 9 We define controllers f^{inv} and $f^{\emptyset}$ with respect to LTS T and invariance property P as: $f^{inv}(q) := A_{inv}(q)$ and $f^{\emptyset}(q) := \emptyset$ for any $q \in Q$.

The controller f^{inv} selects the invariant control actions of each state as its decision whereas $f^{\emptyset}$ disables every action. We can show that f^{inv} is the least tolerant controller whereas $f^{\emptyset}$ is the most tolerant among all invariant controllers.

Theorem 2 *Let an LTS T and an invariance property P be given. For any invariant controller f with respect to T and P, it follows that $\lceil f^{inv} \rceil \subseteq \lceil f \rceil \subseteq \lceil f^{\emptyset} \rceil$.*

Proof It follows from $F^{inv}(q) \subseteq F(q) \subseteq F^{\emptyset}$ for any invariant controller f where F^{inv}, $F^{\emptyset}$, and F are defined as in Theorem 1 for controllers $f^{\emptyset}$, f^{inv}, and f, respectively. □

Intuitively, controller $f^{\emptyset}$ blocks the system from executing any action regardless of the perturbation. For this reason, $f^{\emptyset}$ provides the largest tolerance set at the trade-off of blocking any run to be generated. On the other hand, controller f^{inv} allows the maximum possible set of runs of T that do not violate property P. Consequently, f^{inv} is more susceptible to perturbations and provides the smallest tolerance set at the trade-off of allowing more behavior to be generated. We provide a more thorough study on this trade-off in the next section.

6 Synthesis of tolerant and permissive controllers

6.1 Permissiveness

Permissiveness measures the "restrictiveness" of the controller with the given LTS, i.e., the behavior of the controlled system $T|f$ (Cassandras and Lafortune 2021). Although we could define permissiveness based on $Runs(T|f)$, this definition would omit possible perturbations in the system, see Remark 4. Since every invariant controller tolerates perturbation set $\Omega = Q_{inv} \times Act \times Q_{inv}$, we define permissiveness based on $Runs(T_{\Omega}|f)$. Thus, permissiveness is defined based on the perturbed environment similar to the definition in Takai (2004).

Definition 10 Given invariant controllers f_1 and f_2 for LTS T and invariance property Q_{inv}, we say that f_1 is more permissive than f_2, denoted by $f_2 \subseteq f_1$, if $Runs(T_{\Omega}|f_2) \subseteq Runs(T_{\Omega}|f_1)$. We write $f_1 \equiv f_2$ when we have $Runs(T_{\Omega}|f_2) = Runs(T_{\Omega}|f_1)$.

Remark 4 Defining permissiveness over $Runs(T_{\Omega}|f)$ also allows a finer comparison of controllers. There might be many controllers that generate the same runs in $T|f$, but they, in general, will generate different runs in $T_{\Omega}|f$.

Example 9 We return to the Example 8, where we defined controller f: $f(q) = \{b\}$ if $q \in Q_{inv}$ and $f(3) = \emptyset$. We define another invariant controller: $f'(1) = f(2) = \{b\}$ and $f'(3) = f'(4) = \emptyset$. Since $f'(q) \subseteq f(q)$, it follows that f is more permissive than f'. Note that $Runs(T|f') = Runs(T|f)$, but $Runs(T_{\Omega}|f') \subset Runs(T_{\Omega}|f)$.

6.2 Pareto optimality

We want to identify controllers that cannot be more permissive without losing tolerance and vice-versa, i.e., identify the Pareto front of this trade-off. First, we formally characterize Pareto optimality with respect to permissiveness and tolerance.

Definition 11 Let an LTS T and an invariance property Q_{inv} be given. An invariant controller f_1 and perturbation set d_1 such that $T_{d_1}|f_1 \models Q_{inv}$ is *Pareto optimal* if there does not exist invariant controller f_2 and perturbation d_2 such that:

$$T_{d_2}|f_2 \models Q_{inv} \;\wedge\; f_1 \subseteq f_2 \;\wedge\; d_1 \subseteq d_2 \;\wedge\; (f_1 \not\equiv f_2 \;\vee\; d_1 \neq d_2)$$

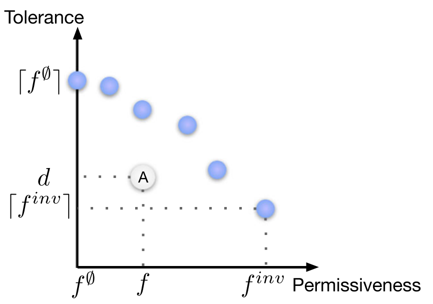
Intuitively, the pair (f_1, d_1) *is not* Pareto optimal if we can improve permissiveness without compromising tolerance or vice-versa. Figure 5a helps us understand Pareto optimality, where the points in blue are Pareto optimal. Note that we identify controllers f^{inv} and $f^{\emptyset}$ in Fig. 5a due to Theorem 2. Based on this result, we establish the first Pareto optimal pairs.

Proposition 1 *The pairs* $(f^{inv}, \lceil f^{inv} \rceil)$ *and* $(f^{\emptyset}, \lceil f^{\emptyset} \rceil)$ *are Pareto optimal.*

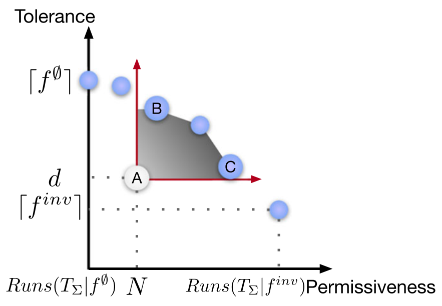
Proof By Theorem 2, $f^{\emptyset}$ and f^{inv} are the most and the least tolerant controllers. We can then show that we cannot modify their permissiveness without compromising tolerance and vice-versa. □

6.3 Synthesis of Pareto controllers

The focus of this section is on synthesizing controllers that generate Pareto optimal pairs, i.e., *Pareto optimal controllers*. Specifically, we want to synthesize Pareto optimal controllers that achieve a desired minimum level of permissiveness and tolerance. Figure 5b helps us explain the investigated problem. Note that, the permissiveness axis is defined based on $Runs(T_{\Omega})$. The desired minimum permissiveness and tolerance are given by a set of runs N and a perturbation set d. Intuitively, we search for a controller f^* that has permissiveness at least N, has tolerance at least d, and $(f^*, \lceil f^* \rceil)$ is Pareto optimal, i.e., $(f^*, \lceil f^* \rceil)$ is a blue point within the shaded region in Fig. 5b. Formally, the problem is stated as follows.



(a) Points in blue are considered the Pareto points. Point A is not Pareto optimal since there exists a point directly above of A that is more tolerant and has the same permissiveness as A. For simplicity, the permissiveness and tolerance axes are simplified as linear even though they are partially ordered sets.



(b) Point A defines the minimum tolerance d and permissiveness N. The solution space of the synthesis problem is depicted in the shaded region. Blue points are considered Pareto points. Points B and C are the most tolerant and most permissive points, respectively.

Fig. 5 Pictorial explanation of Pareto optimality and Problem 3

Problem 3 Given an LTS T, an invariance property Q_{inv}, an set of runs $N \subseteq Runs(T_\Omega)$, and perturbation d, synthesize controller f^*, if it exists, such that (i) $(f^*, \lceil f^* \rceil)$ is Pareto optimal; and (ii) $d \subseteq \lceil f^* \rceil$ and $N \subseteq Runs(T_\Omega | f^*)$.

Problem 3 might not have, in general, a solution for any given sets d and N. For example, if a run in N visits a state outside of Q_{inv}, i.e., it violates the invariance property, then Problem 3 does not have a solution. In the case where a solution exists, Problem 3 might not, in general, have a unique solution. Therefore, we focus on two solutions for this problem, points B and C in Fig. 5b. Point B describes the most tolerant controller within the solution space. To obtain this solution, we add a third condition to ensure we obtain the most tolerant controller. Formally, we have the following problem:

Problem 4 Given an LTS T, an invariance property Q_{inv}, a set of runs $N \subseteq Runs(T_\Omega)$, and a perturbation d, synthesize controller f^*, if it exists, such that (i) $(f^*, \lceil f^* \rceil)$ is Pareto optimal; (ii) $d \subseteq \lceil f^* \rceil$ and $N \subseteq Runs(T_\Omega | f^*)$; and *(Tol)* $\forall f'$ that satisfies (i) and (ii), $\lceil f' \rceil \subseteq \lceil f^* \rceil$.

On the other hand, point C defines the most permissive controller within the solution space. Again, we add a third condition to ensure we obtain the most permissiveness controller. Formally, we have the following problem:

Problem 5 Given an LTS T, an invariance property Q_{inv}, a set of runs $N \subseteq Runs(T_\Omega)$, and a perturbation d, synthesize controller f^*, if it exists, such that (i) $(f^*, \lceil f^* \rceil)$ is Pareto optimal; (ii) $d \subseteq \lceil f^* \rceil$ and $N \subseteq Runs(T_\Omega | f^*)$; and *(Perm)* $\forall f'$ that satisfies (i) and (ii), $f' \subseteq f^*$.

Remark 5 In Problem 3, we assume that the set of runs N is given. It is also possible to assume that instead of N, we are given an invariant controller f that achieves this set of runs, i.e., the set N can be defined as $N := Runs(T_\Omega | f)$.

6.4 Memoryless controllers

We already established that the pairs $(f^{inv}, \lceil f^{inv} \rceil)$ and $(f^{\emptyset}, \lceil f^{\emptyset} \rceil)$ are Pareto optimal. Now, we focus on identifying other Pareto optimal controllers. We start by showing that every memoryless invariant controller and their tolerance form a Pareto optimal pair.

Lemma 3 *Let an LTS T, an invariance property Q_{inv}, and an invariant controller f be given. If f is memoryless, then the pair $(f, \lceil f \rceil)$ is Pareto optimal.*

Proof We prove the theorem by contradiction. Assume that f is invariant and memoryless and that $(f, \lceil f \rceil)$ is not Pareto optimal. It means that $\exists (f_1, d_1)$ such that $T_{d_1} | f_1 \models Q_{inv} \wedge f \subseteq f_1 \wedge \lceil f \rceil \subseteq d_1 \wedge (f \not\equiv f_1 \vee \lceil f \rceil \neq d_1)$.

Since $f \subseteq f_1$, it follows from Definition 7 that $\lceil f_1 \rceil \subseteq \lceil f \rceil$. As we assume that $\lceil f \rceil \subseteq d_1$, it must be that $\lceil f_1 \rceil \subseteq d_1$. Since $\lceil f_1 \rceil$ is the largest tolerable perturbation with respect to f_1, the equality $d_1 = \lceil f_1 \rceil = \lceil f \rceil$ holds.

As $d_1 = \lceil f_1 \rceil = \lceil f \rceil$, we have that $F(q) = F_1(q)$ for any $q \in Q$, where F and F_1 are defined as in Definition 7 for f and f_1, respectively. The equality of functions F and F_1 and the fact that f is memoryless imply that $Runs(T_\Omega | f_1) \subseteq Runs(T_\Omega | f)$, i.e., $f_1 \subseteq f$. As we assumed that $f \subseteq f_1$, we have the equality $f \equiv f_1$, which results in a contradiction of the third clause in the definition of Pareto optimality. □

According to Lemma 3, $(f, \lceil f \rceil)$ is Pareto optimal when f is memoryless and invariant. Controllers with memory and their level of tolerance, in general, *do not* form a Pareto optimal pair. Ideally, we would like to show the converse of Lemma 3, i.e., memoryless controllers are the only Pareto optimal points. However, the generality of our controller definition prevents us making such a claim. For example, controller f defined as $f(q) = \emptyset$ for $q \in I$ and $f(x_{0\ldots n}) = A_{inv}(x_n)$ for $x_{0\ldots n} \in Q^+ \setminus I$ and its tolerance $\lceil f \rceil$ form a Pareto optimal pair equivalent to $(f^{\emptyset}, \lceil f^{\emptyset} \rceil)$, i.e., $\lceil f \rceil = \lceil f^{\emptyset} \rceil$ and $f \equiv f^{\emptyset}$; and f has, in general, memory. Thus, controllers f and $f^{\emptyset}$ with their tolerance generate the same Pareto optimal pair.

We show that there always exists a memoryless controller with its tolerance that generates the same Pareto optimal pair as a controller with memory. In other words, given a Pareto optimal pair $(f, \lceil f \rceil)$, we can always define a memoryless controller f_m such that $f \equiv f_m$ and $\lceil f \rceil = \lceil f_m \rceil$. If controller f is already memoryless, then f_m and f are identical. On the other hand, if f has memory, then f_m flattens the memory used in f.

Lemma 4 *Consider an LTS T, an invariance property Q_{inv}, and an invariant controller f. We define the memoryless controller f_m as follows*

$$f_m(q) := \bigcup_{\substack{x_{0\ldots n} \in \Omega \\ q = x_n}} f(x_{0\ldots n})$$

If $(f, \lceil f \rceil)$ is Pareto optimal, then $(f_m, \lceil f_m \rceil)$ is also Pareto optimal with $f \equiv f_m$ and $\lceil f_m \rceil = \lceil f \rceil$.

Proof We prove the theorem by a direct proof. By the definition of f_m, the functions F and F_m defined as in Definition 7 based on f and f_m, respectively, are equal, i.e., $F(q) = F_m(q)$ for any $q \in Q$. Therefore, we have that $\lceil f \rceil = \lceil f_m \rceil$. Again by the definition of f_m, it follows that $Runs(T_\Omega | f) \subseteq Runs(T_\Omega | f_m)$. As $(f, \lceil f \rceil)$ is assumed to be Pareto optimal, then $Runs(T_\Omega | f_m) \subseteq Runs(T_\Omega | f)$. Therefore, we have that $f \equiv f_m$. □

While not being the converse of Lemmas 3, 4 is an important sufficient result. Combining these two lemmas tells us that memoryless controllers are sufficient for Pareto optimality when considering Pareto pairs of the type $(f, \lceil f \rceil)$.

Theorem 3 *Let an LTS T and an invariance property Q_{inv} be given. It is sufficient to search among invariant memoryless controllers for Pareto optimal pairs of the type $(f, \lceil f \rceil)$.*

Remark 6 Since invariance properties are defined by partitioning the LTS state set, memory does not provide any additional information to controllers to satisfy an invariance property. Similarly, memory does not provide any leverage to violate an invariance property. For this reason, we can show that memoryless controllers are sufficient to describe the Pareto optimal pair of the type $(f, \lceil f \rceil)$.

6.5 Existence of controllers

As we mentioned earlier, Problem 3 might not have a solution. In this section, we provide necessary and sufficient conditions for the existence of solutions. Thanks to Theorem 3, we can focus on memoryless controllers.

The existence of a solution to Problem 3 mainly depends on sets N and d. The first condition for the existence of a solution states that the runs in N do not violate the invariance property Q_{inv}. The second condition checks that it is feasible to generate N and have tolerance d by constructing a controller f that minimally ensures the runs in N and checking if f is invariant and $d \subseteq \lceil f \rceil$. We start by defining this controller that minimally encompasses N.

Definition 12 Assume the premises of Problem 3. We define memoryless controller f^N for each state $q \in Q$ as follows:

$$f^N(q) := \{a \in Act \mid [(\exists x_0 a_0 \ldots x_n \in N \wedge i < n).(a = a_i \wedge q = x_i)\} \cup \{a \in Act \mid [(\exists x_0 a_0 \cdots \in N \wedge i \geq 0)].(a = a_i \wedge q = x_i)\}$$

Intuitively, the controller f^N ensures $N \subseteq Runs(T_\Omega | f^N)$ by construction. Moreover, it ensures this condition by only including actions used in N, i.e., the above construction is a minimal construction in the sense that f^N disconsidered actions not used in N. Based on f^N, we can state the necessary and sufficient conditions for the existence of a solution to Problem 3.

Theorem 4 *Problem 3 has a solution if and only if (a) the runs in N remain in Q_{inv}, (b) f^N is an invariant controller, and (c) $d \subseteq \lceil f^N \rceil$.*

Proof (Only if) Assume that f^* is a solution to Problem 3. We show that (a), (b), and (c) hold. As f^* is a solution to Problem 3, it follows that $N \subseteq Runs(T_\Omega | f^*)$ and N remains in Q_{inv}. Next, we show that $Runs(T_\Omega | f^N) \subseteq Runs(T_\Omega | f^*)$ by contradiction. Let us assume that $Runs(T_\Omega | f^N) \nsubseteq Runs(T_\Omega | f^*)$, which provides us a run $\rho \in Runs(T_\Omega | f^N) \setminus Runs(T_\Omega | f^*)$. By construction of f^N, ρ is either a run or a prefix of a run in N. In either case, it follows that $N \nsubseteq Runs(T_\Omega | f^*)$, which contradicts our assumption that f^* is a solution to Problem 3. Based on the definitions of invariant controller and tolerance together with $Runs(T_\Omega | f^N) \subseteq Runs(T_\Omega | f^*)$, we can conclude that f^N is an invariant controller and $d \subseteq \lceil f^N \rceil$.

[If] We show that f^N is a solution to Problem 3 when (a), (b), and (c) hold. Condition (b) provides that f^N is invariant. By construction of f^N and condition (a), we have that f^N is memoryless as well as $N \subseteq Runs(T_\Omega | f^N)$. Lastly, condition (c) provides that $d \subseteq \lceil f^N \rceil$. □

Example 10 The LTS T is depicted in Fig. 3a and $Q_{inv} = \{1, 2, 4\}$. We want to synthesize a controller that reaches all three states without any perturbation, i.e., $N = \{1a4b2b1\}$. Moreover, we want a controller that is tolerant against the following perturbation set $d = \{(2, a, 3)\}$. The only run in N stays within Q_{inv} which satisfies condition (a) in Theorem 4. Following Theorem 4, we construct f^N: $f^N(1) = \{a\}$, $f^N(2) = f^N(4) = \{b\}$, and $f^N(3) = \emptyset$. It follows that $\lceil f^N \rceil = (Q \times Act \times Q) \setminus \{(1, a, 3), (2, b, 3), (4, b, 3)\}$ and $d \subseteq \lceil f^N \rceil$. Therefore, Problem 3 has a solution.

If we considered perturbation set $d' = \{(2, a, 3), (2, b, 3)\}$, then Problem 3 has no solution. In this latter scenario, the sets N and d' disagree in state 2 with respect to action b, i.e., d' disallows the use of action b in state 2 while N allows this action.

Beyond the existence conditions, Theorem 4 also provides a solution to Problem 3 when a solution exists. Controller f^N is a solution to Problem 3 when the conditions in Theorem 4 are satisfied. We develop this result in the next section since f^N is a special solution to Problem 3.

6.6 Tolerant and permissive controllers

Our first solution to Problem 3 is a controller that is the most tolerant within the solution space (point B in Fig. 5b), i.e., a solution to Problem 4. We show that controller f^N defined in Definition 12 is a solution to Problem 4.

Theorem 5 *Assume that Problem 3 has a solution. Controller f^N is a solution to Problem 4.*

Proof The proof of Theorem 4 already shows that f^N is a solution to Problem 3. Therefore, we just need to show that *(Tol)* holds. In the proof of Theorem 4, we have shown that $Runs(T_\Omega | f^N) \subseteq Runs(T_\Omega | f')$ for any f' that is a solution to Problem 3. Therefore, it follows that $\lceil f' \rceil \subseteq \lceil f^N \rceil$ for any f' that is a solution to Problem 3. □

Example 11 Let us return to the premises in Example 10. We want to synthesize the most tolerant controller that satisfies $N = \{1a4b2b1\}$ and $d = \{(2, a, 3)\}$. Theorem 5 guarantees that controller f^N is the most tolerant controller that satisfies Problem 3, where $f^N(1) = \{a\}$, $f^N(2) = f^N(4) = \{b\}$, and $f^N(3) = \emptyset$. This controller has tolerance $\lceil f^N \rceil = (Q \times Act \times Q) \setminus \{(1, a, 3), (2, b, 3), (4, b, 3)\}$.

The second solution to Problem 3 is the most permissive controller within the solution space (point C in Fig. 5b), i.e., a solution to Problem 5. Based on Theorem 3, we can restrict our attention to memoryless controllers to obtain a solution to Problem 5. Our solution strategy is to augment T with the minimum tolerance required in Problem 5, i.e., analyze the system T_d. The next step is to define the most permissive controller with respect to T_d.

Theorem 6 *Controller f^d defined as $f^d(q) := \{a \in Act \mid Post_{T_d}(q, a) \subseteq Q_{inv}\}$ is a solution to Problem 5.*

Proof Since f^d is memoryless, then $(f^d, \lceil f^d \rceil)$ is Pareto optimal. Next, we show that $d \subseteq \lceil f^d \rceil$ by a direct proof. For any T and invariant controller f, we have that $R \subseteq \lceil f \rceil$. Controller f^d is an invariant controller with respect to T_d and Q_{inv}, i.e., $T_d | f^d \models Q_{inv}$ and all actions of f^d are invariant actions. Thus, we have that $d \subseteq \lceil f^d \rceil$.

We show that $N \subseteq Runs(T_\Omega | f^d)$ by contradiction. Assume that $N \nsubseteq Runs(T_\Omega | f^d)$, then there exists a run in $N \setminus Runs(T_\Omega | f^d)$. This run also belongs to $Runs(T_\Omega | f^N)$ by definition of f^N. It follows that in some state q, f^N can take an action that f^d cannot.

Based on the definition of f^d, we have that $d \nsubseteq \lceil f^N \rceil$, which contradicts our assumption. Therefore, it must be that $N \subseteq Runs(T_\Omega | f^d)$.

We establish that *(Perm)* holds by contradiction. Assume that there exists an invariant controller f' that is a solution to Problem 3 and $f^d \subseteq f'$. We can establish that every action f^d can take f' can also take, but there is an action that f' takes but f^d does not. By the definition of f^d, it follows that f' is not an invariant controller with respect to T_d and $d \nsubseteq \lceil f' \rceil$. It must be that f^d satisfies condition *(Perm)*. □

Example 12 Let us return to the premises in Example 10. We want to synthesize the most permissive controller that satisfies $N = \{1a4b2b1\}$ and $d = \{(2, a, 3)\}$. Theorem 6 guarantees that controller f^d is the most permissive controller that satisfies Problem 3, where $f^d(1) = \{a, b\}$, $f^d(2) = f^d(4) = \{b\}$, and $f^d(3) = \emptyset$. This controller has tolerance $\lceil f^d \rceil = (Q \times Act \times Q) \setminus \{(1, a, 3), (1, b, 3), (2, b, 3), (4, b, 3)\}$.

Controller f^d differs from controller f^N in state 1. In state 1, the most permissive controller allows actions a and b while the most tolerant only allows action a. In this manner, the run $1b1b1b\ldots$ is feasible in the controlled system $T|f^d$, but it cannot be executed in the controlled system $T|f^N$. On the other hand, f^N is tolerant against perturbation $(1, b, 3)$ while f^d is not.

6.7 Complexity analysis

In this section, we analyze the computational complexity of our solutions to Problems 3-5. We summarize our results in Table 1.

To obtain these results, we assume that the set of runs N is represented as an LTS. With abuse of notation, we say that LTS N defined by $\langle Q_N, Act, R_N, I_N \rangle$ generates the desired level of permissiveness. Note that the runs of N are defined over states Q_N while runs of T_Ω are defined over states Q. Therefore, we assume that we are given a mapping from Q_N to Q to circumvent this problem. This mapping can be obtained by composing LTSs T_Ω and N, i.e., $T_\Omega || N$ where $||$ is the standard parallel composition operator (Cassandras and Lafortune 2021). Based on LTS N, we provide the complexity analysis of our results.

Checking the conditions in Theorem 4 has worst-case complexity $O(|Q_N| + |R_N| + |Q|^2)$ due to the construction of f^N and $\lceil f^N \rceil$. Both constructions are reducible to reachability analysis over N and T_Ω. Similarly, the worst-case effort to compute the most tolerant controller, Theorem 5, is $O(|Q_N| + |R_N|)$. Lastly, $f^d(d)$, Theorem 6, can be computed by executing a one step transition in the relation $R \cup d$. Therefore, computing f^d has $O(|Q| + |R \cup d|)$ worst-case complexity.

7 Case studies

In this section, we demonstrate the utility of the proposed notion of tolerance through case studies on the surveillance example described in Section 2 as well as on a simple robot motion

Table 1 Summary of synthesis results

	Existence	Most tol. Point B	Most perm. Point C
Theorem	Theorem 4	Theorem 5	Theorem 6
Complexity	$O(\lvert Q_N\rvert + \lvert R_N\rvert + \lvert Q\rvert^2)$	$O(\lvert Q_N\rvert + \lvert R_N\rvert)$	$O(\lvert Q\rvert + \lvert R \cup d\rvert)$

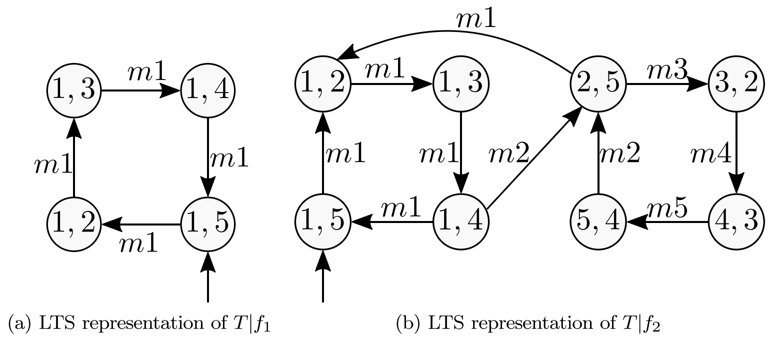


Fig. 6 *Ego* under control of f_1 and f_2

planning scenario as described in Topcu et al. (2012). We have built a prototype tool[2] that can compute the tolerance of a given controller (Algorithm 2) and synthesize controllers $f^{\emptyset}$, f^{inv}, f^N and f^d with respect to invariance properties using the MDESops library (Meira-Góes et al. 2017). We also evaluate the scalability of our tool by comparing it against the brute force Algorithm 1 for computing the tolerance of a controller. The experiments for the case studies were performed on a Linux Ubuntu 20.04 LTS OS machine with 3.2GHz CPU and 32GB memory.

7.1 Surveillance example

In our first case study, we demonstrate how our tool can be used to automatically compute tolerance for different controllers, and how this information can be used to systematically compare alternative controller designs with respect to their tolerance. We also evaluate the scalability of our tool.

7.1.1 Models and property

Example 1 describes how the surveillance example is modeled as an LTS. The invariance property is defined as $Q_{inv} = Q \setminus \{(2,2),(3,3),(4,4),(5,5)\}$. Next, we define two controllers, f_1 and f_2, that satisfy this invariance property. First, we consider controller f_1 to be the one described in Example 2 where it maintains *ego* in location 1. Another controller f_2 ensures that *ego* visits all locations without being captured by *srv*. Formally, f_2 is defined as follows: $f_2(q) = f_1(q)$ if $q \in Q \setminus \{(1,4),(2,5)\}$, $f_2(1,4) = \{m_1, m_2\}$, and $f_2(2,5) = \{m1, m3\}$. Figure 6 shows the LTS representations of $T|f_1$ and $T|f_2$.

7.1.2 Computing the tolerance

We use our tool to compute the tolerance for both controllers f_1 and f_2. Note that LTS T has 20 states, 5 actions, 60 transitions, and the safe set Q_{inv} has 16 states. The tolerance $\lceil f_1 \rceil$ has 1936 transitions of which 1876 are new transitions with respect to the transition relation R. On the other hand, $\lceil f_2 \rceil$ has 1928 transitions where 1868 are new transitions. In both cases, it takes about $8ms$ to compute the tolerance. In comparison, the most tolerant controller $f^{\emptyset}$ characterized by $\lceil f^{\emptyset} \rceil = Q \times Act \times Q$ has 2000 transitions, i.e., the transition relation is complete. The least tolerant controller f^{inv} has tolerance $\lceil f^{inv} \rceil$ with 1716 transitions.

[2] https://github.com/romulo-goes/tolerancetool

7.1.3 Comparing controllers

Controllers f_1 and f_2 select the same control decisions in all states except in states $(1, 4)$ and $(2, 5)$. In these two states, controller f_2 allows *ego* to venture closer to the building. Therefore, controller f_1 should be more tolerant than controller f_2. This intuition is confirmed by our notion of tolerance where $\lceil f_2 \rceil \subset \lceil f_1 \rceil$, i.e., controller f_1 tolerates more perturbations than f_2. Even though f_1 is more tolerant, it is also less permissive as it prevents the *ego* drone from traveling to certain locations that f_2 allows.

7.1.4 Performance analysis

To evaluate the performance of Algorithm 2, we scale the surveillance example by adding more locations as well as more surveillance drones. Table 2 summarizes the evaluation of our tool. The tolerances $\lceil f^{inv} \rceil$ in each of these examples are almost the complete transition relation, i.e., $\lceil f^{inv} \rceil \approx Q \times Act \times Q$. Our tool ran out of memory and it could not compute the tolerance for the system with 10 locations and 3 srv drones; the complete transition relation for this system has 262 144 000 transitions. As part of future work, we plan to improve our tool by symbolic encoding of the LTS, e.g., using OBDD (Bryant 1992). We also compare Algorithm 2 against the brute force Algorithm 1. We implement Algorithm 1 leveraging FuseIC3 (Dureja and Rozier 2017), a state-of-the-art tool that can be used to verify a family of LTS. FuseIC3 more efficiently verifies every possible pertubed system $T_d|f$ that satisfies an invariance property P (lines 2-4 in Algorithm 2. Since the brute force algorithm verifies $T_d|f \models P$ for every perturbation $d \subseteq Q \times Act \times Q$, it is infeasible to use the surveillance example as there are 2^{1940} systems to verify. For this reason, we make this comparison using a modified version of the LTS shown in Fig. 3a. Table 3 summarizes the results of our comparison, which shows that our tool provides a more efficient way of computing tolerance. To be fair, this is not surprising, as although FuseIC3 efficiently verifies a large family of LTS, it was not developed to solve Problem 2. On the other hand, Algorithm 2 directly computes the tolerance of the LTS by leveraging the results of Theorem 1.

7.2 Robot motion planning

In this second case study, we demonstrate how our tool can be used to synthesize a controller that meets a minimum required level of tolerance and permissiveness as specified by the designer.

7.2.1 Models and property

We use the robot motion planning described in Topcu et al. (2012). Consider a robot in an $n \times n$ grid that has three different control actions, $Act = \{M, R, L\}$, which correspond to

Table 2 Scalability of tolerance computation

System	$\|Q\|$	$\|Act\|$	$\|R\|$	$\|\lceil f^{inv} \rceil\|$	time
1 srv, 5 loc.	20	5	60	1 716	0.01 sec
1 srv, 10 loc.	80	10	272	59 030	0.46 sec
2 srv, 10 loc.	640	10	2 176	3 618 978	30.88 sec
3 srv, 10 loc.	5120	10	17 408	out of memory	– sec

Table 3 Comparison with FuseIC3

$\|Q\|$	$\|Act\|$	$\|R\|$	# perturbations	Algorithm 2	Algorithm 1 with FuseIC3
4	2	22	2^{10}	**0.001 sec**	1.5 sec
4	2	17	2^{15}	**0.001 sec**	48.1 sec

move straight, turn right, and turn left, respectively. The state space of the robot is composed of its x, y grid coordinates and its $\theta \in \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$ heading angle. The transition relation is given by the following equations:

– Action = L: $x' = x$, $y' = y$, and $\theta' = \theta + \frac{\pi}{2}$;
– Action = R: $x' = x$, $y' = y$, and $\theta' = \theta - \frac{\pi}{2}$;
– Action = M: $x' = x + \cos(\theta)$, $y' = y + \sin(\theta)$, and $\theta' = \theta$.

where x', y', θ' describe the next state values. We introduce an additional state Out to indicate when the robot goes out of the grid. We set the initial state to be in the center of the grid, i.e., $q_0 = \{(\lfloor\frac{n}{2}\rfloor, \lfloor\frac{n}{2}\rfloor, 0)\}$ where $\lfloor.\rfloor$ is the floor operator. The invariance property is defined by $Q_{inv} = Q \setminus \{Out\}$, i.e., the robot cannot go out of the grid.

7.2.2 Perturbation and permissive sets

To demonstrate our tolerant synthesis framework, we introduce perturbations for actions L, R, M that must be tolerated by the synthesized controller. This perturbation set is constructed based on the following equations:

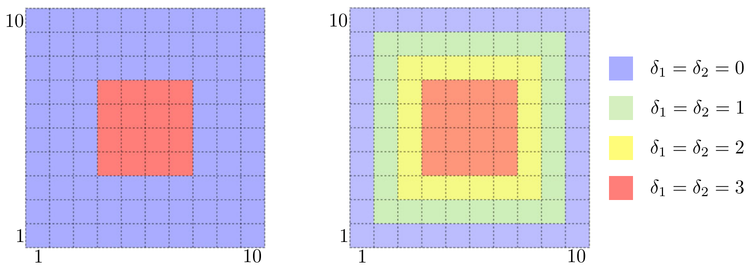
– Action = L: $x' = x + \delta_x\sqrt{2}\cos(\theta + \frac{\pi}{4})$, $y' = y + \delta_y\sqrt{2}\sin(\theta + \frac{\pi}{4})$, and $\theta' = \theta + \frac{\pi}{2}$, $\delta_x, \delta_y \in \{0, 1, \ldots, \delta_1\}$
– Action = R: $x' = x + \delta_x\sqrt{2}\cos(\theta - \frac{\pi}{4})$, $y' = y + \delta_y\sqrt{2}\sin(\theta - \frac{\pi}{4})$, and $\theta' = \theta - \frac{\pi}{2}$, $\delta_x, \delta_y \in \{0, 1, \ldots, \delta_1\}$
– Action = M: $x' = x + (1 + \delta_x)\cos(\theta)$, $y' = y + (1 + \delta_y)\sin(\theta)$, and $\theta' = \theta$, $\delta_x, \delta_y \in \{0, 1, \ldots, \delta_2\}$

Here, δ_1, δ_2 are nonnegative numbers that represent different levels of perturbations. The value δ_1 captures perturbations when the robot turns, i.e., the robot might not stay in the same cell when it turns. Similarly, δ_2 allows the robot to move beyond adjacent grid positions when it moves straight. Again, we use the state Out to indicate when the robot goes out of the $n \times n$ grid. The set $d(\delta_1, \delta_2)$ denotes the set of perturbed transitions defined by the equations above.

With respect to permissiveness, we define desired set of runs N based on regions of the grid we want the robot to explore. Intuitively, we want the robot being able to explore every state in $4 \le x \le 7$ and $4 \le y \le 7$. Pictorially, the robot must be able to visit at least the red region in Fig. 7a. Formally, the set N is defined as $N = \{(x_0, y_0, \theta_0)\ldots(x_m, y_m, \theta_m) \in T_\Omega | (\forall i \le m)[4 \le x_i \le 7 \wedge 4 \le y_i \le 7]\}$.

7.2.3 Synthesizing Pareto controllers

Based on Problem 3, we synthesize Pareto controllers for different desired perturbation sets $d(\delta_1, \delta_2)$ and set of runs N. For illustration, we choose $n = 10$ and synthesize Pareto controllers f^N as defined in Theorem 5 and $f^{d(\delta_1,\delta_2)}$ as defined in Theorem 6.



(a) Pictorial representation of set N, i.e. the robot's desired reachable areas.

(b) Representation of the robot's reachable areas under controllers $f^{d(\delta_1,\delta_2)}$, i.e., the reachable states of the controlled system $T|f^{d(\delta_1,\delta_2)}$. The reachable regions are described by overlapping concentric squares.

Fig. 7 Robot motion planning reachable areas

With respect to the most tolerant controller, f^N, we can synthesize a single controller that is tolerant against perturbations $d(\delta_1, \delta_2)$ for $0 \leq \delta_1 \leq 3$ and $0 \leq \delta_2 \leq 3$. With f^N, the robot can reach exactly the red region in Fig. 7a. In the case of $\delta_1 = \delta_2 = 4$, Problem 3 does not accept solutions as N and $d(\delta_1, \delta_2)$ provide inconsistent requirements. For example, while N requires the robot to visit state $(4, 4, 0)$, a disturbance in $d(4, 4)$ can push the robot outside of the grid region. For this reason, there is no controller that achieves the minimum levels of tolerance $d(4, 4)$ and permissiveness N.

With respect to the most permissive controllers $f^{d(\delta_1,\delta_2)}$, when $\delta_1 = \delta_2 = 0$, the set $d(0, 0)$ is exactly the same as the set R, i.e., no perturbation. For this reason, the synthesized controller $f^{d(0,0)}$ can reach every state in the $n \times n$ grid as depicted in Fig. 7b which satisfies the permissiveness requirement N. Intuitively, the robot can go to the border of the grid since it assumes that turns only alter the heading angle and moves only go to adjacent cells. When $\delta_1 = \delta_2 = 1$, the synthesized controller needs to be more restrictive since moves can go beyond the adjacent cells and turns can alter the robot's position. In this case, $f^{d(1,1)}$ restricts the robot to stay one cell away from the border of the grid as precaution based on the possible perturbations in $d(1, 1)$. In similar manner, the other controllers $f^{d(\delta_1,\delta_2)}$ will restrict the robot's reachable area based on the perturbations while satisfying the permissiveness constraint N. Figure 7b depicts how restrictive these controllers are by showing the reachable areas for each controller.

When $\delta_1 \neq \delta_2$, the reachable states in $T|f^{d(\delta_1,\delta_2)}$ are the same as in $T|f^{d(\max\{\delta_1,\delta_2\},\max\{\delta_1,\delta_2\})}$, e.g., $T|f^{d(0,1)}$ reaches the same states as $T|f^{d(1,1)}$. The difference in these controllers only appears when we analyze the perturbed systems. For example, controller $f^{d(1,1)}$ is blocking in state $(1, 2, \pi)$ since it cannot turn as perturbations $\big((1, 2, \pi), L, Out\big)$ and $\big((1, 2, \pi), R, Out\big)$ belong to $d(1, 1)$. Controller $f^{d(0,1)}$ is nonblocking $(1, 2, \pi)$ since the same perturbations do not belong to $d(0, 1)$.

For $n = 10$, the time to compute controllers $f^{d(\delta_1,\delta_2)}$ varies from $82ms$ when $f^{d(1,1)}$ to $320ms$ when $f^{d(5,5)}$. Table 4 summarizes the results of computing $f^{d(\delta_1,\delta_2)}$ for various n.

8 Related work

Several works have investigated notions of robustness, tolerance, and resilience for discrete transition systems by quantifying perturbation via cost functions, metrics, etc. (Bloem et al.

Table 4 Scalability of computing tolerant controllers

n	$\|Q\|$	$\|Act\|$	δ_1	δ_2	time
10	401	3	3	3	0.21 sec
20	1601	3	3	3	2.87 sec
50	10001	3	3	3	106.70 sec

2014, 2009; Chaudhuri et al. 2011; Henzinger et al. 2014; Majumdar et al. 2011, 2013; Neider et al. 2020; Samanta et al. 2013; Samuel et al. 2020; Tabuada et al. 2012; Filiot et al. 2020). Most of these works define robustness closely to the notions of robustness in continuous state-space control systems, e.g., Bounded-Input-Bounded-Output, (Bloem et al. 2014, 2009; Chaudhuri et al. 2011; Henzinger et al. 2014; Majumdar et al. 2011, 2013; Tabuada et al. 2012; Samanta et al. 2013). Perturbations are either known or they are unknown but with known metric functions, e.g., cost to transition to a different state. Our notion of tolerance is qualitative as it captures the set of perturbations for which the controller guarantees the property and avoids the need for external cost functions over the discrete transition system. In Filiot et al. (2020), perturbations are also introduced based on a cost function over regular languages. The authors investigate the problem of synthesizing the largest threshold that guarantees a system to satisfy a given regular property. Although their synthesis problem is similar to our definition of tolerance, their definition assumes a given metric whereas our tolerance definition is qualitative.

In Neider et al. (2020); Samuel et al. (2020), perturbations are also interpreted as additional transitions to a nominal system. In Neider et al. (2020), a framework to synthesize optimally resilient controllers based on a metric defined by the number of transitions needed to violate an ω-property. Their optimal controller synthesis is similar to our synthesis of tolerant controllers when considering only invariance properties. However, our work also considers permissiveness as part of our controller synthesis problem for invariance properties. In Samuel et al. (2020), the authors propose an abstract-based controller design where the discrete abstraction system is computed with the addition of known perturbations called W_{high}. Compared to our work, the W_{high} perturbations introduced a perturbation set to the finite state abstraction. An optimal controller is then designed based on techniques from Neider et al. (2020).

With respect to qualitative robustness notions, the work in Topcu et al. (2012) investigated synthesizing controllers robust against perturbation sets specified by the designer. The framework of Topcu et al. (2012) does not address the computation of the tolerance of controllers, and as a result does not provide tools to compare the relative tolerance of different controllers such as controllers 1 and 2 in our motivating example. In Tabuada and Neider (2016), the authors presented the notion of robust linear temporal logic (rLTL) which extends the binary view of LTL to a 5-valued semantics to capture different levels of property satisfaction. This work is tangent to ours as it focuses on specifying robustness.

The notion of robustness presented in Kang (2020); Zhang et al. (2020) is only semantically defined. In Zhang et al. (2020), the environmental perturbation is captured by a set of input traces the software system accepts. Perturbations in Kang (2020) are connected to different attack models for software systems. We define the syntax of perturbations as additional transitions in the environment model. Building on our tolerance definition, we have investigated our tolerance definition for general safety properties in Meira-Góes et al. (2023). However, the definition of tolerance in Meira-Góes et al. (2023) differs from Definition 7 since it directly compares transitions instead of the runs of the perturbed system. Moreover, computing tolerance for general safety properties has exponential-time complexity in Meira-Góes et al. (2023). Algorithm 2, on the other hand, has quadratic-time complexity for invariance properties.

There also exists a vast literature on robust control in discrete event systems (Alves et al. 2019; Cury and Krogh 1999; Lin 1993; authorname 2014; Lin et al. 2019; Meira-Góes et al. 2019; Meira-Goes et al. year; Meira-Góes et al. 2022; Paoli and Lafortune 2005; Rohloff 2012; Takai 2004; Wang et al. 2016; Young and Garg 1995). The notions of robustness in Alves et al. (2019); authorname (2014); Lin et al. (2019); Meira-Góes et al. (2019); Meira-Goes et al. (year); Meira-Góes et al. (2022); Rohloff (2012); Wang et al. (2016) are specific to communication delays, loss of information, or deception attacks. Our notion of tolerance represents general model uncertainty, which includes unreliable communication channels in the controlled system.

Of particular relevance to this paper are the works in Cury and Krogh (1999); Lin (1993); Takai (2004), where robustness against model uncertainty is tackled in the context of supervisory control theory. Given a set of plants, Lin (1993) describes methods to find a supervisor that satisfies a property for all plants. Cury and Krogh (1999) focus on synthesizing the most robust supervisor with respect to perturbations in a nominal plant. Extending the work in Cury and Krogh (1999), Takai (2004) investigates the trade-off between permissiveness and robustness in the context of supervisory control theory. Thus, the work in Takai (2004) resembles to our discussion of permissiveness versus tolerance. However, the semantic of our work differs from Takai (2004) as we use a different modeling formalism. We consider state-based controllers that react based on state history, while Takai (2004) considers controllers that react to action history.

The description of the general algorithm to compute $\Delta(T, f, P)$ for any property P connects our work to the work on verifying software product lines (SPL) described as *feature transition systems* (FTS) (Classen et al. 2010, 2013). However, verifying FTS has exponential worst-case time complexity even for invariance properties whereas our method has linear-time complexity. Modal transition systems (MTS) (Larsen and Thomsen 1988; Huth et al. 2001) can also be used to describe a family of LTS. In D'Ippolito et al. (2012), a controller realizability problem is studied for an environment modeled by an MTS, where a controller satisfies a property in all, some, or none of the LTS family. Our notion of controller explicitly computes which systems in the LTS family satisfy the property.

Finally, related to this paper is the work on fault-tolerance. Fault-tolerance has been studied in the context of distributed systems (Gärtner 1999; Lynch 1996; Pease et al. 1980). The work in Bonakdarpour and Kulkarni (2008); Cheng et al. (2011); Ebnenasir et al. (2008); Girault and Rutten (2009) focuses on the synthesis of fault-tolerant programs by retrofitting initial fault-intolerant programs. These works focus on specific types of fault models, whereas our tolerance notion upper-bounds the perturbations (faults) the controller tolerates.

9 Conclusion

In this paper, we introduced a new notion of tolerance against environmental perturbations for discrete-state control systems. We provided a general technique to compute this tolerance for general properties modeled as regular languages over finite strings as well as a more efficient technique specifically for invariance properties. We also investigated the problem of synthesizing an invariant controller that achieves a given minimum threshold of tolerance and permissiveness. We used Pareto optimality to capture the trade-off between tolerance and permissiveness and showed that memoryless controllers are sufficient to capture the Pareto front of this trade-off.

Our notion of tolerance is syntactically defined by additional transitions and semantically defined by the controlled behavior generated by these additional transitions. However, the additional transitions and new controlled behavior need to be analyzed by a designer as to explain them. We leave to future work to bridge this gap between the syntax of our notion of tolerance with the context of the model to provide tolerance explanations to the designer. As part of future work, we will also devise more efficient techniques for properties other than invariance. One may also investigate connections between Problem 1 and the solution of two-player Büchi games (Grädel et al. 2002). In Remark 1, we discussed the use of the nondeterministic transition function to model uncontrollable actions. However, we leave to future work to investigate our framework with controllable and uncontrollable actions. It would be also interesting to extend our work to investigate tolerance in the context of partially observable systems. Lastly, the connection between the fields of supervisory control theory and reactive synthesis has been recently investigated (Ehlers et al. 2017; Schmuck et al. 2020). As part of future work, we will investigate if our framework can be extended to the supervisory control theory framework.

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

Appendix

A controller with finite memory M is a triple composed by the control function $f : M \times Q \rightarrow 2^{Act}$, a memory update function $g : M \times Q \rightarrow M$, and an initial memory condition $m_0 \in M$. In this manner, the closed-loop system $T|f$ can be represented by an LTS.

Definition 13 Let an LTS T and a controller (f, g, m_0) with bounded memory M be given. We can define LTS $T|f = (Q \times M, Act, R|f, I \times \{m_0\})$ where $R|f$ is defined as follows:

$$\begin{aligned} R|f := \{(q, m, a, q', m') \in Q \times M \times Act \times Q \times M \mid (q, a, q') \in R \wedge \\ a \in f(m, q) \wedge m' = g(m, q')\} \end{aligned} \tag{1}$$

Due to the memory M introduced by the controller f, $Runs(T|f)$ and $Paths(T|f)$ are defined over $Q \times M \times Act$ and $Q \times M$, respectively. However, the ω-properties are only defined over Q. Therefore, we project $Q \times M \times Act \rightarrow Q \times Act$ and $Q \times M \rightarrow Q$ using the operator $\downharpoonleft$. With an abuse of notation, we use $Runs(T|f)$ and $Paths(T|f)$ to be the projected runs and paths $Runs(T|f) \downharpoonleft$ and $Paths(T|f) \downharpoonleft$

References

Alur R, La Torre S (2001) Deterministic generators and games for LTL fragments. In: Proceedings 16th annual IEEE symposium on logic in computer science, pp 291–300

Alves MVS, da Cunha AEC, Carvalho LK, Moreira MV, Basilio JC (2019) Robust supervisory control of discrete event systems against intermittent loss of observations. Int J Control, pp 1–13

Baier C, Katoen JP (2008) Principles of Model Checking. The MIT Press

Ball T, Levin V, Rajamani SK (2011) A decade of software model checking with slam. Commun ACM 54(7):68–76

Belta C, Yordanov B, Aydın Göl E (2017) Formal Methods for Discrete-Time Dynamical Systems, 1st edn. Springer Publishing Company

Bloem R, Jobstmann B, Piterman N, Pnueli A, Sa'ar Y (2012) Synthesis of reactive(1) designs. J Comput Syst Sci 78(3):911–938

Bloem R, Chatterjee K, Greimel K, Henzinger TA, Hofferek G, Jobstmann B, Könighofer B, Könighofer R (2014) Synthesizing robust systems. Acta Inf 51(3–4):193–220

Bloem R, Greimel K, Henzinger TA, Jobstmann B (2009) Synthesizing robust systems. In: 2009 Formal methods in computer-aided design, pp 85–92

Bonakdarpour B, Kulkarni SS (2008) Sycraft: a tool for synthesizing distributed fault-tolerant programs. In: van Breugel F, Chechik M (eds) CONCUR 2008 - Concurrency Theory. Springer, Berlin Heidelberg, pp 167–171

Bryant RE (1992) Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Comput Surv 24(3):293–318

Cassandras CG, Lafortune S (2021) Introduction to discrete event systems, 3rd edn. Springer, Cham

Chaudhuri S, Gulwani S, Lublinerman R, Navidpour S (2011) Proving programs robust. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on foundations of software engineering, ESEC/FSE '11. Association for Computing Machinery, pp 102–112

Cheng CH, Rueß H, Knoll A, Buckl C (2011) Synthesis of fault-tolerant embedded systems using games: from theory to practice. In: Jhala R, Schmidt D (eds) Verification, model checking, and abstract interpretation. Springer, Berlin Heidelberg, pp 118–133

Classen A, Cordy M, Schobbens PY, Heymans P, Legay A, Raskin JF (2013) Featured transition systems: foundations for verifying variability-intensive systems and their application to LTL model checking. IEEE Trans Softw Eng 39(8)

Classen A, Heymans P, Schobbens PY, Legay A, Raskin JF (2010) Model checking lots of systems: efficient verification of temporal properties in software product lines. In: 2010 ACM/IEEE 32nd International conference on software engineering, vol 1, pp 335–344

Cury J, Krogh B (1999) Robustness of supervisors for discrete-event systems. IEEE Trans Autom Control 44(2):376–379

D'Ippolito N, Braberman V, Piterman N, Uchitel S (2012) The modal transition system control problem. In: Giannakopoulou D, Méry D (eds) FM 2012: formal methods. Springer, Berlin Heidelberg, pp 155–170

Dureja R, Rozier KY (2017) FuseIC3: an algorithm for checking large design spaces. In: 2017 Formal methods in computer aided design (FMCAD), pp 164–171

Ebnenasir A, Kulkarni SS, Arora A (2008) FTSyn: a framework for automatic synthesis of fault-tolerance. Int J Softw Tools Technol Transf 10(5):455–471

Ehlers R, Lafortune S, Tripakis S, Vardi MY (2017) Supervisory control and reactive synthesis: a comparative introduction. Discrete Event Dynamic Systems 27:209–260

Filiot E, Mazzocchi N, Raskin JF, Sankaranarayanan S, Trivedi A (2020) Weighted transducers for robustness verification. In: Konnov I, Kovács L (eds) 31st International conference on concurrency theory (CONCUR 2020), Leibniz International Proceedings in Informatics (LIPIcs), vol 171, pp 17:1–17:21. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany

Gärtner FC (1999) Fundamentals of fault-tolerant distributed computing in asynchronous environments. ACM Comput Surv 31(1):1–26

Girault A, Rutten E (2009) Automating the addition of fault tolerance with discrete controller synthesis. Formal Methods in System Design 35:190–225

Grädel E, Thomas W, Wilke T (eds) (2002) Automata logics, and infinite games: a guide to current research. Springer-Verlag, Berlin, Heidelberg

Henzinger TA, Otop J, Samanta R (2014) Lipschitz robustness of finite-state transducers. In: Raman V, Suresh SP (eds) 34th International conference on foundation of software technology and theoretical computer science (FSTTCS 2014), Leibniz International Proceedings in Informatics (LIPIcs), vol 29, pp 431–443. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany,

Huth M, Jagadeesan R, Schmidt D (2001) Modal transition systems: a foundation for three-valued program analysis. In: Sands D (ed) Programming languages and systems. Springer, Berlin Heidelberg, pp 155–169

Kang E (2020) Robustness analysis for secure software design. In: Proceedings of the 3rd ACM SIGSOFT international workshop on software security from design to deployment, SEAD 2020, p 19–25. Association for Computing Machinery

Larsen K, Thomsen B (1988) A modal process logic. In: 1988 Proceedings. Third annual symposium on logic in computer science, pp 203–210

Lin F (1993) Robust and adaptive supervisory control of discrete event systems. IEEE Trans Autom Control 38(12):1848–1852

Lin F (2014) Control of networked discrete event systems: dealing with communication delays and losses. SIAM J Control Optim 52(2):1276–1298

Lin L, Zhu Y, Su R (2019) Towards bounded synthesis of resilient supervisors. In: 2019 IEEE 58th Conference on Decision and Control (CDC), pp 7659–7664

Lynch NA (1996) Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

Majumdar R, Render E, Tabuada P (2011) Robust discrete synthesis against unspecified disturbances. In: Proceedings of the 14th International conference on hybrid systems: computation and control, HSCC '11, p 211–220. Association for Computing Machinery

Majumdar R, Render E, Tabuada P (2013) A theory of robust omega-regular software synthesis. ACM Trans Embed Comput Syst 13(3)

Meira-Goes R, Lafortune S, Marchand H (2021) Synthesis of supervisors robust against sensor deception attacks. IEEE Trans Autom Control 66(10):4990–4997

Meira-Góes R, Dardik I, Kang E, Lafortune S, Tripakis S (2023) Safe environmental envelopes of discrete systems. In: (to appear) Computer aided verification

Meira-Góes R, Kang E, Lafortune S, Tripakis S (2022) On synthesizing tolerable and permissive controllers for labeled transition systems. In: 16th IFAC Workshop on discrete event systems WODES 2022

Meira-Góes R, Marchand H, Lafortune S (2019) Towards resilient supervisors against sensor deception attacks. In: 2019 IEEE 58th Annual conference on decision and control (CDC)

Meira-Góes R, Marchand H, Lafortune S (2022) Dealing with sensor and actuator deception attacks in supervisory control. Automatica

Meira-Góes R, Wintenberg A, Matsui S, Lafortune S (2017) MDESops: an open-source software tool for discrete event systems modeled by automata. In: Proc 22nd IFAC World Congr

Neider D, Weinert A, Zimmermann M (2020) Synthesizing optimally resilient controllers. Acta Inf 57(1):195–221

Newcombe C, Rath T, Zhang F, Munteanu B, Brooker M, Deardeuff M (2015) How amazon web services uses formal methods. Commun ACM 58(4):66–73

Paoli A, Lafortune S (2005) Safe diagnosability for fault-tolerant supervision of discrete-event systems. Automatica 41(8):1335–1347

Pease M, Shostak R, Lamport L (1980) Reaching agreement in the presence of faults. J ACM 27(2):228–234

Pnueli A (1977) The temporal logic of programs. In: 18th Annual symposium on foundations of computer science (sfcs 1977), pp 46–57

Pnueli A, Rosner R (1989) On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL '89, pp 179–190. Association for Computing Machinery

Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25(1):206–230

Rohloff K (2012) Bounded sensor failure tolerant supervisory control. 11th IFAC Workshop on Discrete Event Systems 45(29):272–277

Samanta R, Deshmukh JV, Chaudhuri S (2013) Robustness analysis of string transducers. In: Van Hung D, Ogawa M (eds) Automated technology for verification and analysis. Springer Publishing Company, pp 427–441

Samuel S, Mallik K, Schmuck AK, Neider D (2020) Resilient abstraction-based controller design. In: 2020 59th IEEE conference on decision and control (CDC), pp 2123–2129

Schmuck AK, Moor T, Majumdar R (2020) On the relation between reactive synthesis and supervisory control of non-terminating processes. Discrete Event Dynamic Systems 30(1):81–124

Tabuada P (2009) Verification and control of hybrid systems: a symbolic approach, 1st edn. Springer Publishing Company

Tabuada P, Neider D (2016) Robust linear temporal logic. In: Talbot JM, Regnier L (eds) 25th EACSL Annual conference on computer science logic (CSL 2016), Leibniz International Proceedings in Informatics (LIPIcs), vol 62, pp 10:1–10:21. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany

Tabuada P, Balkan A, Caliskan SY, Shoukry Y, Majumdar R (2012) Input-output robustness for discrete systems. In: Proceedings of the tenth ACM international conference on embedded software, EMSOFT '12, p 217–226. Association for Computing Machinery

Takai S (2004) Maximizing robustness of supervisors for partially observed discrete event systems. Automatica 40(3):531–535

Topcu U, Ozay N, Liu J, Murray RM (2012) On synthesizing robust discrete controllers under modeling uncertainty. In: Proceedings of the 15th ACM international conference on hybrid systems: computation and control, HSCC '12, pp 85–94. Association for Computing Machinery

Wang F, Shu S, Lin F (2016) Robust networked control of discrete event systems. IEEE Trans Autom Sci Eng 13(4):1528–1540

Young S, Garg VK (1995) Model uncertainty in discrete event systems. SIAM J Control Optim 33(1):208–226
Zhang C, Garlan D, Kang E (2020) A behavioral notion of robustness for software systems. In: Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering, ESEC/FSE 2020, pp 1–12. Association for Computing Machinery

Rômulo Meira-Góes is an Assistant Professor in the Department of Electrical Engineering at Pennsylvania State University. Previously, he was a postdoctoral researcher working with Eunsuk Kang, Stavros Tripakis and Stéphane Lafortune. In 2022, he received the inaugural CPS rising stars award from UVA. He received his Ph.D. in Electrical and Computer Engineering from the University of Michigan in 2020, working with Stéphane Lafortune. Prior to the University of Michigan, he earned his B.S. degree in Electrical Engineering from the Universidade Tecnológica Federal do Paraná - Curitiba in 2015. His research interests include supervisory control of discrete event systems, formal methods and game theory, specially, their application in cyber security of cyber-physical systems.

Eunsuk Kang is an Assistant Professor in the Software and Societal Systems Department, School of Computer Science at Carnegie Mellon University. He received a Ph.D. in Electrical Engineering and Computer Science from MIT, and a Bachelor of Software Engineering from the University of Waterloo in Canada. His research interests include software engineering, formal methods, security, and system safety.

Stéphane Lafortune was born in Montréal, Québec, Canada. He received the B.Eng degree from École Polytechnique de Montréal in 1980, the M.Eng degree from McGill University in 1982, and the Ph.D degree from the University of California at Berkeley in 1986, all in electrical engineering. Since September 1986, he has been with the University of Michigan, Ann Arbor, where he is a Professor of Electrical Engineering and Computer Science. In March 2018, he was appointed as the N. Harris McClamroch Collegiate Professor of Electrical Engineering and Computer Science. Lafortune is a Fellow of the IEEE (1999) and of IFAC (2017). He received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994 (for a paper co-authored with S.-L. Chung and F. Lin) and in 2001 (for a paper co-authored with G. Barrett). Lafortune's research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer and software systems. He co-authored, with C. Cassandras, the textbook Introduction to Discrete Event Systems (Third Edition, Springer, 2021). He also authored the book A Guide to Signals and Systems in Continuous Time (Springer 2022). Lafortune served as Editor-in-Chief of the Journal of Discrete Event Dynamic Systems: Theory and Applications from 2015 to 2020.

Stavros Tripakis is an Associate Professor of Computer Science at Northeastern University. He received his Ph.D. degree in Computer Science at the Verimag Laboratory, Joseph Fourier University, Grenoble, France, and has held positions at the University of California at Berkeley, at the French National Research Center CNRS, at Cadence Design Systems, and at Aalto University. His research interests are in the foundations of software and system design, formal verification, and cyber-physical systems. Dr. Tripakis was co-Chair of the 10th ACM & IEEE Conference on Embedded Software (EMSOFT 2010), and Secretary/Treasurer (2009-2011) and Vice-Chair (2011-2013) of ACM SIGBED. His H-index is 54.

Authors and Affiliations

Rômulo Meira-Góes[1] · **Eunsuk Kang**[2] · **Stéphane Lafortune**[3] · **Stavros Tripakis**[4]

Eunsuk Kang
eskang@cmu.edu

Stéphane Lafortune
stephane@umich.edu

Stavros Tripakis
stavros@northeastern.edu

1 School of Electrical Engineering and Computer Science, The Pennsylvania State University, University Park, PA, USA

2 School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

3 Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

4 Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA