Fault-Tolerant Supervisory Control

Thomas Moor *

* *Lehrstuhl für Regelungstechnik*
Friedrich-Alexander Universität Erlangen-Nürnberg, Germany
(e-mail: lrt@fau.de)

Abstract: A system is *fault tolerant* if it remains functional after the occurrence of a fault. Given a plant subject to a fault, *fault-tolerant control* requires the controller to form a fault tolerant closed-loop system. For the systematic design of a fault-tolerant controller, typical input data consists of the plant dynamics including the effect of the faults under consideration and a formal performance requirement with a possible allowance for degraded performance after the fault. For its obvious practical relevance, the synthesis of fault-tolerant controllers has received extensive attention in the literature, however, with a particular focus on continuous-variable systems. This paper provides an overview on the synthesis of fault-tolerant controllers within the framework of supervisory control to address discrete-event systems that are adequately represented by regular languages. [*Revised version — June 2015*]

Keywords: discrete-event systems, supervisory control, fault-tolerant control, passive fault-tolerant control, active fault-tolerant control, post-fault recovery, fault-hiding approach.

INTRODUCTION

Following the general introduction given in Blanke et al. (2006), a fault is considered a sudden change in the behaviour of a system with potentially negative consequences to the overall performance regarding predefined objectives. When it comes to control, the system consists of a plant and a controller. A common setting here is to require the controller to compensate the fault to some degree in order to maintain an operational closed loop and to achieve a possibly degraded but well defined overall performance. Such a controller is termed *fault tolerant*. This paper discusses the synthesis of fault tolerant control for discrete-event systems that are adequately representable by formal languages.

In contrast to continuous systems, where the sudden change of behaviour requires a suitable extension of the modelling framework, discrete-event systems can represent the occurrence of faults seamlessly. On the other hand, the modelling procedure for typical applications of discrete-event systems is fundamentally different, in that the model is derived from a textual description of the plant and its environment rather than directly from physical principles. It is therefore not straight forward to formalise what an acceptable degraded performance may be for a particular application. Thus, while the above plain human language definition of fault tolerance applies to both domains, the challenges in formal methods for verification and synthesis of fault-tolerance systems differ substantially.

This paper is meant to accompany an invited talk at the *5th International Workshop on Dependable Control of Discrete-Event Systems (5th IFAC DCDS 2015), Mexico,* to provide an overview — it does not contain any original contribution. For a homogeneous and concise notation, a purely language based framework for the control of discrete-event systems (Section 1) is used as a basis to discuss *passive fault tolerance* (Section 2), *active fault tolerance* (Section 3), *post-fault recovery* (Section 4), and an adaption of the *fault-hiding approach* (Section 5), with the overall scope restricted to deterministic discrete-event systems. Observing applicable constraints, the paper covers selected approaches provided by the literature, and the reader is kindly invited to follow the given references for the more detailed original expositions.

PRELIMINARIES AND NOTATION

This section provides notation and elementary facts on formal languages as relevant for the present paper. For a general introduction see (Hopcroft and Ullman, 1979), and, for a discrete-event systems perspective, (Cassandras and Lafortune, 2008).

Let Σ be a *finite alphabet*, i.e., a finite set of symbols $\sigma \in \Sigma$. The *Kleene-closure* Σ^* is the set of finite strings $s = \sigma_1\sigma_2\cdots\sigma_n$, $n \in \mathbb{N}$, $\sigma_i \in \Sigma$, and the *empty string* $\epsilon \in \Sigma^*$, $\epsilon \notin \Sigma$. The length of a string $s \in \Sigma^*$ is denoted $|s| \in \mathbb{N}_0$, with $|\epsilon| = 0$. If, for two strings $s, r \in \Sigma^*$, there exists $t \in \Sigma^*$ such that $s = rt$, we say r is a *prefix* of s, and write $r \leq s$; if in addition $r \neq s$, we say r is a *strict prefix* of s and write $r < s$. The prefix of $s \in \Sigma^*$ with length $n \in \mathbb{N}_0$, $n \leq |s|$, is denoted $\mathrm{pre}_n\, s$. In particular, $\mathrm{pre}_0\, s = \epsilon$ and $\mathrm{pre}_{|s|}\, s = s$. If, for two strings $s, t \in \Sigma^*$, there exists $r \in \Sigma^*$ such that $s = rt$, we say t is a *suffix* of s. The suffix of a string $s \in \Sigma^*$ obtained by deleting the prefix of length n, $n \leq |s|$, is denoted $\mathrm{suf}_n\, s$; i.e., $s = (\mathrm{pre}_n\, s)(\mathrm{suf}_n\, s)$, $\mathrm{suf}_0\, s = s$ and $\mathrm{suf}_{|s|}\, s = \epsilon$.

A **-language* (or short a *language*) over Σ is a subset $L \subseteq \Sigma^*$. Given a language $L \subseteq \Sigma^*$, the equivalence relation $[\equiv_L]$ on Σ^* is defined by $s'\,[\equiv_L]\,s''$ if and only if $(\forall\, t \in \Sigma^*)[\, s't \in L \leftrightarrow s''t \in L\,]$. The language L is *regular* if $[\equiv_L]$ has only finitely many equivalence classes, and, thus is accepted by a finite automaton.

The *prefix* of a language $L \subseteq \Sigma^*$ is defined by $\mathrm{pre}\, L := \{r \in \Sigma^* \mid \exists\, s \in L : r \leq s\}$. The prefix operator distributes over arbitrary unions of languages. However, for the intersection of two languages L and K, we have $\mathrm{pre}\,(L \cap K) \subseteq (\mathrm{pre}\, L) \cap (\mathrm{pre}\, K)$. If equality holds, L and K are said to be *non-conflicting*. This is trivially the case for $K \subseteq L$. The prefix operator is also referred to as the *prefix-closure*, and, a language L is *closed* if $L = \mathrm{pre}\, L$.

A language K is *relatively closed w.r.t. L* if $K = (\mathrm{pre}\, K) \cap L$. The intersection $\mathrm{pre}\,(K) \cap L$ is always relatively closed w.r.t. L. If a language K is relatively closed w.r.t. a closed language, then K itself is closed.

For two languages $L,\ M \subseteq \Sigma^*$, the *concatenation* is defined $LM := \{\, st \mid s \in N,\ t \in M\}$. The concatenation of closed languages is closed. The *relative suffix* is defined $L/M := \{\, t \mid \exists\, s \in M : st \in L\,\}$. If L is closed, so is L/M. For two languages $K,\ M \subseteq \Sigma^*$, K is said to *converge asymptotically* to M, denoted by $M \leftarrow K$, if for each $s \in K$, there exists an i such that $\mathrm{suf}_i\, s \in M$. This is equivalent to $K \subseteq \Sigma^* M$. Moreover, K is said to *converge finitely* to M, denoted by $K \Leftarrow M$, if there is a non-negative integer n such that for each $s \in K$, there exists an $i \leq n$ such that $\mathrm{suf}_i\, s \in M$. If $M \Leftarrow K$, the least possible n is called the *convergence time*; see also (Willner and Heymann, 1995). Finite convergence with convergence time n implies $K \subseteq (\cup_{i \leq n} \Sigma^i) M$. The latter inclusion is also proposed by Kumar et al. (1993) to define the notion of *language stability*.

For the *observable events* $\Sigma_o \subseteq \Sigma$, the *natural projection* $\mathrm{p_o}\colon \Sigma^* \to \Sigma_o^*$ is defined iteratively: (1) let $\mathrm{p_o}\epsilon := \epsilon$; (2) for $s \in \Sigma^*$, $\sigma \in \Sigma$, let $\mathrm{p_o}(s\sigma) := (\mathrm{p_o} s)\sigma$ if $\sigma \in \Sigma_o$, or, if $\sigma \notin \Sigma_o$, let $\mathrm{p_o}(s\sigma) := \mathrm{p_o} s$. The set-valued inverse $\mathrm{p_o^{-1}}$ of $\mathrm{p_o}$ is defined by $\mathrm{p_o^{-1}}(r) := \{s \in \Sigma^* \mid \mathrm{p_o}(s) = r\}$ for $r \in \Sigma_o^*$. When applied to languages, the projection distributes over unions, and the inverse projection distributes over unions and intersections. The prefix operator commutes with projection and inverse projection.

The *synchronous composition* of two languages L_1 and L_2 over Σ_1 and Σ_2, respectively, is defined by $L_1 \parallel L_2 := (\mathrm{p_1^{-1}} L_1) \cap (\mathrm{p_2^{-1}} L_2)$, where $\mathrm{p_1}$ and $\mathrm{p_2}$ denote the natural projections from $\Sigma = \Sigma_1 \cup \Sigma_2^*$ to Σ_1^* and Σ_2^*, respectively. Here, L_1 and L_2 are said to be *non-conflicting*, if $(\mathrm{pre}\, L_1) \parallel (\mathrm{pre}\, L_2) = \mathrm{pre}\,(L_1 \parallel L_2)$. For $\Sigma_1 = \Sigma_2$ the synchronous composition amounts to language intersection. For $\Sigma_1 \cap \Sigma_2 = \emptyset$ the synchronous composition is also called the *shuffle product*.

Given two languages $L,\ K \subseteq \Sigma^*$, and a set of *uncontrollable events* $\Sigma_{uc} \subseteq \Sigma$, we say K is *controllable w.r.t. L*, if $(\mathrm{pre}\, K)\Sigma_{uc} \cap (\mathrm{pre}\, L) \subseteq \mathrm{pre}\, K$. Note that, in contrast to e.g. (Ramadge and Wonham, 1987) but in compliance with e.g. (Cassandras and Lafortune, 2008), this variant of controllability does not insist in $K \subseteq L$. With $\Sigma_o \subseteq \Sigma$ the set of observable events, we say K is *prefix-normal w.r.t. L* (or short *normal w.r.t. L*), if $\mathrm{pre}\, K = (\mathrm{p_o^{-1}} \mathrm{p_o}\, \mathrm{pre}\, K) \cap (\mathrm{pre}\, L)$. A language $K \subseteq \Sigma^*$ is *complete*, if for all $s \in \mathrm{pre}\, K$ there exists $\sigma \in \Sigma$ such that $s\sigma \in \mathrm{pre}\, K$. Each of the properties controllability, normality, completeness, closedness and relative closedness is retained under arbitrary union: given a family of languages $(K_a)_{a \in A}$, $K_a \subseteq L$ for all $a \in A$, such that a particular combination of the mentioned properties is possessed by each K_a, then the union $K := \cup_{a \in A} K_a$ possesses the respective properties, too. Note that closedness and relative closedness are also retained under arbitrary intersection.

Unless otherwise noted, the alphabets Σ, Σ_c, Σ_{uc}, Σ_o and Σ_{uo} refer to the *common partitioning* $\Sigma = \Sigma_c \dot\cup \Sigma_{uc} = \Sigma_o \dot\cup \Sigma_{uo}$ in controllable, uncontrollable, observable and unobservable events, respectively.

1. SUPERVISORY CONTROL

We revisit the basic control problem studied in supervisory control theory as introduced by Ramadge and Wonham (1987), including the further development to account for partial observation by Lin and Wonham (1988), in a variation that turns out convenient for the present paper. The following discussion also includes some comments on how a discrete-event system relates to the respective physical phenomenon under consideration; see also (Cassandras and Lafortune, 2008; Wonham, 1999).

§ 1: Modelling

Following the introduction given in Ramadge and Wonham (1989), we consider phenomena which can be adequately represented with a discrete state set and piece-wise constant state trajectories. Changes in the value of the state variable are referred to as *transitions*. While the state is regarded internal to the phenomenon, individual transitions are labelled with *events* from a finite alphabet Σ to be externally visible when the respective transition occurs. Thus, an observation of the phenomenon for some arbitrarily long but finite duration yields a finite sequence of events, each taking place at a particular instance of physical time. We restrict attention to phenomena where the physical timing is regarded irrelevant and where only the order of events shall be represented by the model. The resulting abstract notion of time is referred to as *logic time*, and any possible outcome of the above observation can be represented as a string $s \in \Sigma^*$ interpreted w.r.t. logic time. To this end, the set $L \subseteq \Sigma^*$ of all possible strings the phenomenon can generate within any arbitrary finite amount of physical time is regarded a *discrete-event system* that models the phenomenon under consideration. A consequence of this definition is that if the phenomenon can generate the string $s \in L$ then it can also generate any prefix $t \leq s$ and, therefore, we have $L = \mathrm{pre}\, L$. We emphasise this fact by denoting our model by $\mathrm{pre}\, L$, referred to as the *local behaviour*. This interpretation of a formal language as a discrete-event system will be refined in due course.

§ 2: Elementary Properties

Informally, a phenomenon possesses a *safety property* if "something bad can not happen". In the proposed modelling framework, this corresponds to a set of bad strings not to be generated. Using a language $E \subseteq \Sigma^*$ to represent the complement of the bad strings, the respective safety property can be stated as set inclusion

$$\mathrm{pre}\, L \subseteq E. \tag{1}$$

Since the local behaviour on the left-hand side of the inclusion is prefix closed, E can be substituted by the supremal prefix-closed sublanguage of E without affecting the imposed constraint. In particular, any safety property can be represented by a prefix-closed upper bound on the local behaviour.

In contrast to safety, a *liveness property* requires that "something good will happen". We recall two liveness properties commonly discussed in the context of $*$-languages. A local behaviour $\mathrm{pre}\, L \subseteq \Sigma^*$ *does not deadlock*, if it can always be extended by one more event, i.e., if [1]

$$(\forall\, s \in \mathrm{pre}\, L)(\exists\, \sigma \in \Sigma)[\, s\sigma \in \mathrm{pre}\, L\,]. \tag{2}$$

To deduce from the above formula a liveness property in the intended sense, we impose an additional assumption on the phenomenon: *if, at any physical time, the phenomenon can generate one more event then the phenomenon will generate one more event*. Then, a non-empty local behaviour that does not deadlock models a phenomenon that generates within infinite

[1] Technically, Eq. (2) can be rephrased as a prefix-closed upper bound on the local behaviour, however, in contrast to the situation of safety properties, the bound here depends on L.

physical time an infinite number of events. Such phenomena are also referred to as *non-terminating processes*. [2]

The second liveness property we recall is parametrised by a set $M \subseteq \Sigma^*$ of strings to indicate positively distinguished configurations of the phenomenon, with *task completion* as the most common interpretation. Here, we refer to M as the *accepted behaviour*. We say that $\operatorname{pre} L \subseteq \Sigma^*$ *does not livelock* w.r.t. M, if [3]

$$(\forall\, s \in \operatorname{pre} L)(\exists\, t \in \Sigma^*)[\, st \in M \cap \operatorname{pre} L\,], \qquad (3)$$

i.e., if there is the persistent possibility to attain an accepted string. To obtain a liveness property in the sense of the intended interpretation, we assume that: *if, at any physical time when no accepted string is generated, the phenomenon has the chance to generate an accepted string, then it will eventually do so*. With this assumption, non-empty local behaviours that do not livelock model phenomena that will within infinite physical time generate a monotone sequence of accepted strings, which is bounded in the case of termination and unbounded otherwise. A phenomenon that does not livelock w.r.t. an accepted behaviour can be modelled as a single language $L \subseteq \Sigma^*$ with associated local behaviour $\operatorname{pre} L$ and associated accepted behaviour L. To indicate this interpretation of a language we will use the terminology of a *discrete-event system* $L \subseteq \Sigma^*$. This is the perspective we take for the remainder of this paper. [4]

Remark. For a phenomenon that does deadlock or livelock w.r.t. some accepted behaviour, the locking can be made explicit by adding a distinguished event and by extending the local behaviour to generate this event in the situation of a lock. With this transformation, the phenomenon can again be formally modelled as a discrete-event system $L \subseteq \Sigma^*$. In subsequent analysis tasks and synthesis tasks, the distinguished event needs to be considered accordingly; e.g., for controller synthesis as discussed in the following sections, the blocking event is flagged uncontrollable and a language inclusion specification must be put in place to require that the controller prevents any occurrence of the blocking event in the formal closed-loop behaviour. Thus, when applied to the actual phenomenon, configurations that could possibly block will not be reached.

§ 3: System Composition

When a phenomenon is composed from multiple components, one seeks to systematically construct an overall model from individual models. For discrete-event systems, it is common to consider the composition by synchronisation of shared events; i.e., an event can only occur at an instance of physical time if it complies with the event sequence generated so far w.r.t. the local behaviour of each individual component. Thus, when the components are modelled by two prefix-closed languages $L_2 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, the parallel composition

$$L := L_1 \parallel L_2 := (\mathrm{p}_1^{-1} L_1) \cap (\mathrm{p}_2^{-1} L_2)\,, \qquad (4)$$

is an adequate model of the overall phenomenon. Applying the same formula to the general case of not-closed languages amounts to the requirement that accepted configurations are attained simultaneously by all components. Depending on the interpretation of liveness w.r.t. the underlying phenomena this implication may or may not be applicable. Alternatively, one can compose separate models for the local behaviour and the accepted behaviour and merge them according to the above remark. Conflicts are then left to be resolved at a later stage, e.g., by designing a suitable controller.

§ 4: Closed-Loop Configuration

For the purpose of control, the alphabet is composed as a disjoint union of *controllable events* and *uncontrollable events*, i.e., $\Sigma = \Sigma_c \dot\cup \Sigma_{uc}$. To this end, we consider the *plant* to be given as a discrete-event system $L \subseteq \Sigma^*$. We assume, that the underlying phenomenon is equipped with some interface to disable any controllable event at any time. In the original literature (Ramadge and Wonham, 1987, 1989), the controller is represented as a *causal feedback map* $f\colon \operatorname{pre} L \to \Gamma$ with the set of *control-patterns* $\Gamma := \{\gamma \mid \Sigma_{uc} \subseteq \gamma \subseteq \Sigma\}$. There, $f(s)$ is the set of events enabled by the controller after the plant has generated s and until the next event is generated. Note that any assumptions imposed on the phenomenon regarding liveness must comply with the proposed mechanism of control.

In order to account for situations where not every event is reported to the controller, Lin and Wonham (1988) distinguish *observable events* and *unobservable events*, i.e., $\Sigma = \Sigma_o \dot\cup \Sigma_{uo}$, and require the controller to apply consistent control patterns after the generation of strings that can not be distinguished by observation. Formally, this amounts to the condition $f(s') = f(s'')$ for any pair of strings s', $s'' \in \Sigma^*$ with $\mathrm{p}_o s'' = \mathrm{p}_o s'$. As it turns out, the general situation of partial-observation adds substantial complexity to controller synthesis problems; see also Cho and Marcus (1989); Yin and Lafortune (2014). For the following, we pragmatically restrict attention to the less involved case $\Sigma_c \subseteq \Sigma_o$, i.e., we assume that the controller can not disable unobservable events.

In this paper, we represent f as a language H and interpret supervision as a form of system composition. To parallel the setting of *non-blocking supervisory control under partial observation* in (Lin and Wonham, 1988), we impose the following conditions on H. [5]

Definition 1. Given an alphabet with the common partition $\Sigma = \Sigma_c \dot\cup \Sigma_{uc} = \Sigma_o \dot\cup \Sigma_{uo}$, a language H is an *admissible controller* for the plant $L \subseteq \Sigma^*$, if

(H0) H is prefix-closed,
(H1) $H\Sigma_{uc} \subseteq H$,
(H2) $H = \mathrm{p}_o^{-1}\mathrm{p}_o H$,
(H3) $(\operatorname{pre} L) \cap (\operatorname{pre} H)$ is complete , and
(H4) L and H are non-conflicting. □

In the above setting, we interpret $K_{loc} := (\operatorname{pre} L) \cap (\operatorname{pre} H)$ as the local closed-loop behaviour obtained by restricting the occurrence of events to those that are simultaneously enabled by both the plant and the controller. Then, $K := K_{loc} \cap L$ amounts to the

[2] For non-terminating processes, the local behaviour can alternatively be represented by the topologically closed ω-language $\lim \operatorname{pre} L$, i.e., the set of all infinite strings with all prefixes in $\operatorname{pre} L$; see also Ramadge (1989).

[3] Technically, Eq. (3) is equivalent to $\operatorname{pre} L \subseteq \operatorname{pre}(M \cap \operatorname{pre} L)$, which is again a prefix-closed upper bound that depends on L.

[4] In the absence livelocks and deadlocks, the phenomenon can be alternatively represented by an ω-language, namely the set $\lim L$ of infinite strings with infinitely many prefixes in L. Technically, and in contrast to $\lim \operatorname{pre} L$, the limit $\lim L$ is in general not topologically closed. This has relevant implications to synthesis problems that explicitly or implicitly refer to the latter limit.

[5] In the original literature, the plant is represented by an automaton, and, thus, can itself be blocking. For this case we need to apply the transformation proposed in the remark of the preceding section. Another subtle difference is that we require the closed loop not only not to live-lock but also not to dead lock. This is motivated by non-terminating processes and is convenient for the discussion of diagnosis later in this paper.

closed-loop behaviour accepted by the plant. In particular, (H4) implies $K_{\mathrm{loc}} \subseteq \operatorname{pre} K$, i.e., the closed loop does not livelock w.r.t. L and $K = L \cap H$ is a discrete-event system that models the synchronous composition of plant and controller. The following theorem relates the slightly different setting used in the present paper to results from the literature and establishes a characterisation of achievable closed-loop behaviour under admissible control.

Theorem 2. Consider an alphabet with the common partition $\Sigma = \Sigma_c \dot\cup \Sigma_{uc} = \Sigma_o \dot\cup \Sigma_{uo}$ with $\Sigma_c \subseteq \Sigma_o$. For a plant $L \subseteq \Sigma^*$ and an admissible controller $H \subseteq \Sigma^*$ let $K = L \cap H$. Then

(K0) K is relatively prefix-closed w.r.t. L,

(K1) K is controllable w.r.t. L,

(K2) K prefix-normal w.r.t. L, and

(K3) K is complete.

Moreover, if $L \neq \emptyset \neq H$, then $K \neq \emptyset$. Vice versa, if K satisfies (K0)-(K3), then there exists an admissible controller H such that $K = L \cap H$. Moreover, if $K \neq \emptyset$, then $H \neq \emptyset$.

Proof. [Outline] Part 1, "trivial cases". $H = \emptyset$ implies $K = \emptyset$, which in turn satisfies (K0)–(K3). Likewise, for $K = \emptyset$ we can choose $H = \emptyset$ to satisfy (H0)-(H4). Exclude trivial cases from now on. Part 2, "(H*)⇒(K*)". (K0) is a consequence of (H0). (H4) amounts to $\operatorname{pre} K = (\operatorname{pre} L) \cap (\operatorname{pre} H)$. This can be used to establish (K1), (K2) and (K3) from (H1), (H2) and (H3), respectively. Part 3, "(K*)⇒(H*)", is established constructively with the candidate $H := \mathrm{p}_o^{-1}\mathrm{p}_o((\operatorname{pre} K)\Sigma_{uc}^*)$. □

§ 5: Controller Synthesis

Given a plant $L \subseteq \Sigma^*$ and an upper bound language inclusion specification $E \subseteq L$, the common controller synthesis problem is to establish an admissible controller $H \subseteq \Sigma^*$ such that $K = L \cap H \subseteq E$. By (K0) this is the case if and only if the local behaviour $\operatorname{pre} K$ is a subset of the supremal closed sublanguage of E, i.e., the language inclusion specification can be equivalently stated as a prefix-closed upper bound on the local closed-loop behaviour, and is, in this sense, a safety specification. A core observation from the literature is that the closed-loop properties are retained under arbitrary union; e.g. (Ramadge and Wonham, 1989) for controllability, (Lin and Wonham, 1988) for normality, and (Kumar et al., 1992) for completeness. Thus, there exists a unique supremal achievable closed-loop behaviour $K^\uparrow$ that satisfies all of the above properties and the language inclusion specification $K^\uparrow \subseteq E \subseteq L$. Clearly, if and only if the supremum $K^\uparrow$ is non-empty, we can extract a corresponding controller $H \neq \emptyset$ with a non-empty closed-loop behaviour. Thus, for practical applications the synthesis problem is solved by procedures that compute a finite representation of $K^\uparrow$. For regular parameters, various such procedures that address specific combinations of closed-loop properties have been proposed, see e.g. (Cho and Marcus, 1989; Brandt et al., 1990; Kumar et al., 1992) as well as (Moor et al., 2012) for the specific situation of the present paper. [6]

[6] As a follow-up to the previous notes on non-terminating processes, and depending on the application at hand, it may turn out appropriate to interpret the closed loop w.r.t. an infinite logical-time axis and consider the limits on the respective $*$-languages. However, for this interpretation, the assumption $E \subseteq L$ amounts to a relevant loss of generality. Technically, it implies that the limit $\lim E$ is relatively closed w.r.t. to the limit $\lim L$. For more detailed considerations in this regard, see (Thistle and Wonham, 1994; Thistle and Lamouchi, 2009) and also Moor et al. (2012).

2. NAIVE FAULT-TOLERANT CONTROL

Compliant with (Blanke et al., 2006), a fault is considered a sudden change in the behaviour of the plant. Thus, to verify or synthesise fault-tolerance in a model based approach, one needs to extend the so called *nominal model* of the fault-free behaviour to a model that accommodates the occurrence of the fault regarding its possible past (optionally interpreted as the cause of the fault) and the possible future (commonly interpreted as the effect of the fault). Typically, one may accept degraded performance after the occurrence of the fault and ideally one will insist in nominal performance up to the occurrence of the fault. Design options include to seek for a single controller that uniformly operates the plant including the fault to enforce the overall control objective (referred to as *passive fault-tolerant control*), or to explicitly detect the fault and then to switch to a prepared controller that guarantees for acceptable post fault behaviour (referred to as *active fault-tolerant control*). Both approaches amount to a switched plant model, while the second approach in addition introduces a switched overall controller.

For continuous-signal systems, both types of switching constitute a potential challenge in verification and synthesis of fault tolerant controllers since they require a careful review of the classical frameworks used for the nominal design. For a fault-tolerant design, one needs to account for the newly introduced discontinuities when the fault switches between plant models, and, for active fault-tolerant control, when diagnosis schemes detect faults and dynamically trigger discontinuities by switching controllers.

In contrast, discrete-event systems by nature can model sudden changes in the behaviour seamlessly. Thus, one may approach verification and synthesis of fault-tolerant controllers by using the same framework as used for the nominal design. The additional effort required to achieve fault-tolerance then appears at the modelling stage. Moreover, the distinction between active and passive fault-tolerant control may affect solution strategies, but, as long as the modelling paradigms are unchanged, active and passive fault-tolerance is not expected to affect the achievable over-all closed-loop behaviour. Following Wittmann et al. (2012), we outline on how to organise verification and synthesis of fault-tolerance in this *naive approach*.

We begin with a nominal closed-loop configuration, consisting of a nominal alphabet denoted Σ_n with the common partitioning, a nominal plant model $L_n \subseteq \Sigma_n^*$, a nominal language inclusion specification $E_n \subseteq L_n$, and an admissible nominal controller $H_n \subseteq \Sigma_n^*$ according to the requirements (H0)–(H4) with resulting closed-loop system $K_n \subseteq E_n \subseteq L_n$.

To accommodate for the fault, we extend the alphabet by a distinguished event $\mathrm{f} \notin \Sigma_n$, i.e., $\Sigma_f := \Sigma_n \dot\cup \{\mathrm{f}\}$, and define the *degraded plant behaviour* $L_d \subseteq \Sigma_f^*$ to specify all possible pasts that may trigger the fault and all possible post-fault behaviour. Thus, we may assume that

$$\operatorname{pre} L_d \cap \Sigma_n^* \subseteq \operatorname{pre} L_n\,, \tag{5}$$

$$L_d \cap \Sigma_n^* = \emptyset\,. \tag{6}$$

The natural candidate for a *fault-accommodating model* is the conjunction

$$L_f = L_n \cup L_d\,, \tag{7}$$

and the above assumptions Eq. (5) and (6) imply

$$\operatorname{pre} L_{\mathrm{f}} = (\operatorname{pre} L_{\mathrm{n}}) \cup (\Sigma_{\mathrm{n}}^{*}\mathrm{f}\Sigma_{\mathrm{f}}^{*} \cap \operatorname{pre} L_{\mathrm{d}}), \tag{8}$$

$$L_{\mathrm{f}} = L_{\mathrm{n}} \cup (((\operatorname{pre} L_{\mathrm{n}})\mathrm{f}\Sigma_{\mathrm{f}}^{*}) \cap L_{\mathrm{d}}), \tag{9}$$

in support of our construction.

From the controller perspective, the fault event f is regarded uncontrollable. In general, f is also regarded unobservable, however, depending on the level of abstraction one also encounters applications where the plant instantly reports any fault by built-in diagnosis. A fault-accommodating language inclusion specification can be set up from the same pattern as the fault-accommodating plant model, i.e.

$$E_{\mathrm{f}} = E_{\mathrm{n}} \cup E_{\mathrm{d}}, \tag{10}$$

subject to assumptions obtained by uniform substitution from Eqs. (5) and (6). Here, the implied consequences ensure that the specification E_{f} up to the occurrence of the fault is not stricter than the the nominal specification E_{n}.

Once fault-accommodating models L_{f} and E_{f} are provided, options are to verify an existing controller H (e.g. the nominal controller H_{n}) regarding fault tolerance or to synthesise a fault-tolerant controller H_{f} from scratch. Both problems can be solved by the procedures from the nominal control problem, but now applied to the input data L_{f} and E_{f}. Note that, even if H_{f} and H_{n} are considered minimal restrictive, the conditions Eq. (5) and (6) do not imply that the pre-fault behaviour of the closed loop matches the nominal closed-loop behaviour. This is because a given plant L_{f} implies that a fault-tolerant controller avoids those pre-fault configurations, from which, in the case of the fault, the post-fault requirements imposed by E_{f} can not be achieved. Obviously, the nominal controller is not subject to this constraint and therefore leads to a potentially less restrictive pre-fault behaviour. This can be regarded inadequate depending on the application at hand. However, such a situation is considered a consequence of the respective input data L_{f} and E_{f}, but not a fundamental limitation of the presented naive approach to fault-tolerant control.

Remark. In the first instance, the presented approach is motivated to handle *persistent faults*, i.e., faults that can only occur once with certain performance degradation accepted from thereon. There is also the option to have a plant in which the fault can occur more than once. This feature is relevant for plants that shall be controlled to perfect recovery in the sense of eventually matching the pre-fault behaviour, including the possibility of the occurrence of the fault; see also Section 4.

Remark. Throughout this text we will continue to consider only one distinguished fault event. In the first instance, this is motivated by a concise notation. As long as no explicit diagnosis is required, and as long as the fault is persistent, there is no practical benefit in distinguishing multiple fault events.

The interpretation of the presented naive approach as *passive* fault-tolerant control is obvious. In general, there is the potential to produce practical solutions even if the fault is not diagnosable. This is expected to be the case if those causes of a fault (in the sense of pre-fault behaviours), that by their post-fault behaviour conflict with applicable conditions for diagnosability, can be prevented by a more restrictive control of the pre-fault behaviour. However, provided that the fault is diagnosable, an interpretation as *active* fault-tolerant control can be based on the composition of a diagnoser and H_{f}: any first transition to an f-certain state can be regarded as a switching to a post-fault controller; see also Section 3.

3. ACTIVE FAULT-TOLERANT CONTROL

Referring to Blanke et al. (2006), active fault-tolerant control is achieved by two measures applied in the context of the nominal closed-loop configuration: first, a diagnosis mechanism is used in order to detect the fault; and, second, after the fault has been detected, the nominal controller is deactivated and an alternative controller is activated to continue to operate the plant. The general benefit of this approach is that the pre-fault behaviour of the closed loop exactly matches the nominal closed-loop behaviour, including heuristic optimisations not formally captured by the nominal control objectives. The crucial challenge of this approach is to detect the fault early enough in order have the chance to achieve prescribed post-fault performance objectives. In the following we report on an adaption of active fault-tolerant control to discrete-event systems originally developed in (Paoli and Lafortune, 2005; Paoli et al., 2008, 2011) and resemble a simplified variant of the core concepts in the context of the present paper.

At the first stage (A), the cited references begin with an automaton representation of the fault-accommodating model $L_{\mathrm{f}} = L_{\mathrm{n}} \cup L_{\mathrm{d}} \subseteq \Sigma_{\mathrm{f}}$ under nominal control H_{n}, i.e., H_{n} is admissible w.r.t. L_{n} and formally extended not to disable the fault event f. For a concise notation, $(\operatorname{pre} L_{\mathrm{f}}) \cap H_{\mathrm{n}}$ is assumed not to deadlock. The authors then require a diagnoser to report any fault before the system violates a prescribed post-fault safety specification. The latter is given as a set of strings $\Psi \subseteq \Sigma_{\mathrm{n}}^{*}$ that must not occur as a substring of the local post-fault behaviour. The existence of a diagnoser suitable for this task is discussed in (Paoli and Lafortune, 2005) and characterised by a property called *safe diagnosability*, derived as an extension from the general study of discrete-event system diagnosis by Sampath et al. (1995).

Definition 3. Given the fault-accommodating plant $L_{\mathrm{f}} \subseteq \Sigma_{\mathrm{f}}^{*}$ with nominal controller H_{n}, let $L = (\operatorname{pre} L_{\mathrm{f}}) \cap H_{\mathrm{n}}$ and define the *diagnosis condition* $\mathcal{D}[w]$ in terms of $w \in \Sigma_{\mathrm{f}}^{*}$ to be true if $L \cap (\mathrm{p}_{\mathrm{o}}^{-1}\mathrm{p}_{\mathrm{o}}w) \subseteq \Sigma_{\mathrm{n}}^{*}\mathrm{f}\Sigma_{\mathrm{n}}^{*}$. Then L is *safe diagnosable* if there exists a non-negative integer k, such that

$$(\forall\, s \in L \cap (\Sigma_{\mathrm{n}}^{*}\mathrm{f}),\ t \in (\Sigma_{\mathrm{n}})^{k}\Sigma_{\mathrm{n}}^{*})[\, st \in \operatorname{pre} L_{\mathrm{f}} \ \Rightarrow\ \mathcal{D}[st]\,], \tag{11}$$

and if, in the above equation, the shortest prefix of a $t \in (\Sigma_{\mathrm{n}})^{k}$ that satisfies $\mathcal{D}[st]$ for a given s does not contain an illegal substring from Ψ. □

The consequence of the second requirement is that the occurrence of the fault will be diagnosed before an illegal string is generated in the post-fault behaviour. Note that the diagnosis stage (A) is stated in terms of the local behaviour $\operatorname{pre} L_{\mathrm{f}}$ and thus cannot incorporate liveness properties other than not to deadlock. This technically justifies the requirement of a uniform upper bound w.r.t. logic time until a fault is diagnosed. To this end, denote $T \subseteq (\operatorname{pre} L_{\mathrm{f}}) \cap H_{\mathrm{n}}$ the set of strings at which a corresponding diagnoser first enters an f-certain state.

At the second stage (B), addressing the time after the fault has been diagnosed, the plant must still be controlled not to exhibit a string that has a substring within the illegal set Ψ. The existence of an appropriate control scheme for this purpose is characterised by a property called *safe controllability*, proposed in (Paoli et al., 2008). Note that, *safe controllability* technically gives priority to stopping the plant in order to avoid illegal substrings, even if this causes a deadlock. This can be accounted for in the subsequent design stage (C).

For the final design stage (C), post-fault-detection controllers are synthesised to take over the plant after diagnosis of the fault, i.e., when the plant has entered a string in T. The synthesis of these controllers is performed on a strategically constructed formal plant model using the common control objectives observability and controllability. The purpose of the strategically constructed formal plant model is that the switching to one of the post-fault-detection controllers is consistent with the observations available to both, the switching mechanism and the respective controllers. Thus, in the reading of the present paper and under the assumption that $\Sigma_c \subseteq \Sigma_o$, one may accumulate the post-fault detection controllers to a formal language H_d to take over the plant from H_n once the observation has passed $T_o := p_o T$. This amounts to the following fault-tolerant controller:

$$H_f := p_o^{-1}\{r\sigma \in p_o H_n \mid (\mathrm{pre}\, r) \cap T_o = \emptyset\} \cup p_o^{-1}\{r\sigma \in p_o H_d \mid (\mathrm{pre}\, r) \cap T_o \neq \emptyset\} \cup \{\epsilon\}\,. \quad (12)$$

.

An alternative approach to design this over-all control scheme is to use a post-fault language inclusion specification E_d such that

$$\mathrm{pre}\, T \subseteq E_d \subseteq \Sigma_f^* - \Sigma_n^* \mathrm{f} \Sigma_n^* \Psi \Sigma_n^*\,, \quad (13)$$

to solve the synthesis problem with input data L_f and E_f, and to verify whether or not the resulting controller H_f qualifies for the extraction of H_d in compliance with the switching defined by Eq. (12). If E_d only imposes restrictions on the behaviour after the fault can be detected, a minimal restrictive solution H_f should include T in the local closed-loop behaviour, and, thus, should turn out admissible. If explicit diagnosis is not required, one may skip the computation of T and directly use an application specific post-fault specification E_d with

$$(\mathrm{pre}\, L_n) \cap H_n \subseteq E_d \subseteq \Sigma_f^* - \Sigma_n^* \mathrm{f} \Sigma_n^* \Psi \Sigma_n^*\,. \quad (14)$$

If the subsequent verification step fails, one concludes that either the safe diagnosability condition is not satisfied or that the post-fault performance objective can not be achieved.

4. POST-FAULT RECOVERY

The fault-tolerant controller design strategies presented so far do recover after the fault in that their post-fault behaviour satisfies a prescribed language inclusion specification. More explicit approaches to recovery have been proposed by relating the long-term behaviour after the fault with the either the nominal behaviour before the fault or the nominal specification. In the following we report on a framework developed in (Wen et al., 2008b,a, 2014) which addresses recovery in terms of language convergence and variations thereof (Kumar et al., 1993; Willner and Heymann, 1995).

We begin our discussion with an adaption of the notion of *weak fault tolerance* from (Wen et al., 2008b). [7]

Definition 4. A fault-accommodating behaviour $K_f \subseteq \Sigma_f^*$, with nominal part $K_n \subseteq K_f \cap \Sigma_n^*$ is *weakly fault-tolerant* if

$$K_f/(\mathrm{pre}\, K_n) \Leftarrow K_f/(\mathrm{pre}\, K_f - \mathrm{pre}\, K_n)\,, \quad (15)$$

$$(\mathrm{pre}\, K_f)/(\mathrm{pre}\, K_n) \Leftarrow (\mathrm{pre}\, K_f)/(\mathrm{pre}\, K_f - \mathrm{pre}\, K_n)\,. \quad (16)$$

□

[7] The cited literature gives a more general definition in terms of possibly blocking automata and derives the respective variant of Eqs. (15) and (16) as equivalent characterisation; see (Wen et al., 2008b) Definition 3 and Theorem 3.

By the right-hand side, the condition addresses strings with a past that exit the nominal behaviour by the fault event. The left-hand side requires the existence of a future that matches the future of some string within the nominal behaviour. Here, the left-hand side terms account for models in which the fault can occur more than once. For persistent faults and if the convergence time is of no concern, one may use $K_n \Leftarrow K_f/(\Sigma_n^* \mathrm{f})$ as an alternative condition.

The above notion of fault tolerance imposes a restriction on the local post fault behaviour and, in this sense, it is a safety specification. For the recovery of liveness properties, Wen et al. (2014) propose the following stronger condition. [8]

Definition 5. A fault-accommodating behaviour $K_f \subseteq \Sigma_f^*$, with nominal part $K_n \subseteq K_f \cap \Sigma_n^*$ is *fault tolerant*, if there exist a non-negative integer k, such that for all $s, t \in \Sigma_f^*$, $|t| \geq k$, with

$$s \in \mathrm{pre}\, K_f - \mathrm{pre}\, K_n\,, \quad st \in \mathrm{pre}\, K_f \quad (17)$$

there exists $u \in \mathrm{pre}\, K_n$ and $v \leq t$, $|v| \leq k$, and

$$K_f/sv = K_f/u \quad (18)$$

□

It is the equality in the second equation, by which the above notion of fault tolerance requires not only the recovery of safety properties but also the recovery of liveness properties.

For an overall design, that task is to find an admissible controller H_f that enforces a language inclusion specification and fault tolerance in the sense of Definition 5 with $K_n := L_n \cap H_f$ and $K_f := L_f \cap H_f$. Regarding optimality, a nearby objective is to maximise the pre-fault behaviour and to minimise the recovery time. However, Wen et al. (2014) demonstrate by example, that a maximal achievable pre-fault behaviour in general does not exist. Therefore, the cited literature further discusses the synthesis problem in terms of automata representations, where, beginning with a realisation of the relevant behaviours and the language inclusion specification, only subautomata are considered. With this restriction, an algorithmic solution is derived for the case that all events are observable.

An alternative approach for recovery is proposed by Sülek and Schmidt (2014), where the authors impose three closed-loop requirements, one for the pre-fault behaviour, one for a transitional phase after the fault, and a language convergence specification for the long-term post-fault behaviour. A problem statement in the setting here is given as follows. [9]

Definition 6. Given a fault-accommodating plant $L_f \subseteq \Sigma_f^*$, with nominal part $L_n \subseteq L_f \cap \Sigma_n^*$, an admissible controller H is *fault tolerant* if it guarantees the following properties for the closed loop $K_f = L_f \cap H$ for the specification parameters $E_n \subseteq \Sigma_n^*$, $E_d \subseteq \Sigma_f^*$ and $E_f \subseteq \Sigma_f^*$:

(P1) $K_f \cap \Sigma_n^* \subseteq E_n$,

(P2) for all $s \in L_f \cap \Sigma_n^* \mathrm{f} \Sigma_n^*$ there is a partition $s = u_1 v_1 u_2 v_2 \cdots u_k v_k \mathrm{f} t$ such that $u = u_1 u_2 \cdots u_k \in \mathrm{pre}\, E_n$ and $v = v_1 v_2 \cdots v_k t \in E_d$,

(P3) $E_f \Leftarrow K_f/(\Sigma_n^* \mathrm{f})$. □

[8] We refer to (Wen et al., 2014), Definition 2, for the above characterisation in terms of languages. For an automata based formulation see (Wen et al., 2008b), Definition 1. We also note that the original literature is technically more general, in that it explicitly accounts for systems that deadlock.

[9] The original literature uses automata representations for the plant model and, thus, accounts for a blocking plant. Moreover, multiple fault events and marking supervisors are addressed.

The second condition requires that if the fault happens at all, then some fraction of the pre-fault string can be reinterpreted according to the specification E_d, while the remaining fraction complies with E_n. Here, E_d is used to require application specific re-initialisation for post-fault operation. Sülek and Schmidt (2014) give a complete and sound algorithm to synthesise fault tolerant controllers in the sense of Definition 6 for the case that all events are observable. It is also demonstrated how the algorithm can be used for a subsequent repair procedure to retain the nominal specification.

Remark. As a general comment on the concept of language convergence and the above variations, note that they impose a uniform bound on the number of events allowed until satisfactory behaviour is attained. This can be regarded inadequate for phenomena where the effect of a fault only shows after a non-uniformly bounded logic-time duration. This is, e.g., expected whenever the overall plant is built from independent components which possess independent liveness properties. However, when dropping the uniform bound, convergence of $*$-languages becomes in general a to weak requirement (e.g. consider a language $K = (M)^*$; without a uniform bound, any $*$-language converges to K). In such situations, the discussion requires explicit or implicit reference to the corresponding ω-languages. For example, one may require $\lim L \subseteq \lim(\Sigma^* K)$ for a non-uniformly bounded convergence. This is again a language inclusion requirement and, if all events are observable, it can be addressed with the approach proposed by Thistle and Wonham (1994). A related problem of state attraction under partial observation is studied in Schmidt and Breindl (2014).

5. FAULT-HIDING APPROACH

With the design strategy of *fault hiding*, one begins with a given fault-accommodating model $L_\mathrm{f} = L_\mathrm{n} \cup L_\mathrm{d} \subseteq \Sigma_\mathrm{f}^*$ and a nominal controller $H_\mathrm{n} \subseteq \Sigma_\mathrm{n}^*$. One then seeks a reconfiguration mechanism, that, once the fault occurred, re-interprets the control action executed by the nominal controller to operate the actual plant L_f. In turn, the feedback provided by the plant L_f is re-interpreted to generate feedback accepted by the nominal controller. The reconfiguration mechanism is meant to pretend nominal plant behaviour to the nominal controller while imposing fault-tolerant control to the actual plant. In particular, the nominal controller remains permanently active in the overall closed-loop configuration. This addresses situations where the nominal controller not only satisfies a formal control objective but also has been optimised by heuristic methods and/or human expertise. This approach is well developed within the context of continuous control; see e.g. (Richter, 2011).

For discrete-event systems, a fault-hiding approach is provided by Wittmann et al. (2013). The synthesis of a reconfiguration mechanism is re-phrased as the computation of an overall closed-loop behaviour K_f for the plant L_f and the inclusion specification $E_\mathrm{f} = E_\mathrm{n} \cup E_\mathrm{d}$, such that a corresponding admissible controller H_f can be composed from the reconfiguration mechanism R and the nominal controller H_n. Sufficient conditions are obtained as given below, where $h\colon \Sigma_\mathrm{n}^* \to \Sigma_\mathrm{v}^*$ encodes a bijective translation from events in Σ_n to distinct *virtual events* in Σ_v, $\Sigma_\mathrm{v} \cap \Sigma_\mathrm{f} = \emptyset$. [10]

Definition 7. Given the models $L_\mathrm{f} \subseteq \Sigma_\mathrm{f}^*$ and $L_\mathrm{n} \subseteq \Sigma_\mathrm{n}^*$ and the specifications $E_\mathrm{n} \subseteq \Sigma_\mathrm{n}^*$ and $E_\mathrm{f} \subseteq \Sigma_\mathrm{f}^*$, construct the supremal nominal controller $H_n^\uparrow \subseteq \Sigma_\mathrm{n}^*$ and denote its virtualisation by $H_v^\uparrow := h(H_n^\uparrow) \subseteq \Sigma_\mathrm{v}^*$. The following conditions are imposed on the a candidate closed loop $K \subseteq L_\mathrm{f} \| H_v^\uparrow$:

(M1) K is controllable w.r.t. $L_\mathrm{f} \parallel H_v^\uparrow$ and the uncontrollable events $\Sigma_\mathrm{uc} \dot\cup h(\Sigma_\mathrm{c})$

(M2) pre K is prefix-normal w.r.t. pre $(L_\mathrm{f} \parallel H_v^\uparrow)$,

(M3) K is relatively closed w.r.t. $L_\mathrm{f} \parallel H_v^\uparrow$

(M4) K is complete,

(M5) K is weakly sensor-event consistent, i.e.,
$$(\forall\, s \in \mathrm{pre}\, K)[\, (\mathrm{p_v} s) h(\Sigma_\mathrm{uc}) \cap (\mathrm{pre}\, h(L_\mathrm{n})) \neq \emptyset \Rightarrow s(\Sigma - h(\Sigma_\mathrm{c}))^* h(\Sigma_\mathrm{uc}) \cap \mathrm{pre}\, K \neq \emptyset\,]$$

(M6) K operates $H_v^\uparrow$ within $h(L_\mathrm{n})$, i.e., $K \subseteq \mathrm{p_v^{-1}}\, \mathrm{pre}\, h(L_\mathrm{n})$.

(M7) K satisfies the inclusion specification $K \subseteq E_\mathrm{f}$. □

If the conditions are satisfied, then the reconfiguration dynamics can be extracted from K by projection $R = \mathrm{p_r}\, K$. Results reported in Wittmann et al. (2013) include admissibility of $R \| h(H_\mathrm{n})$ w.r.t. L_f for *any* solution H_n to the nominal control problem as well as additional admissibility criteria regarding the interconnections of the three individual components L_f, R and H_n. An according synthesis procedure is elaborated in Wittmann (2014). Since the above properties (M1)–(M7) do not depend on the actual nominal controller H_n, the latter does not need to be known in form of a formal language model. This is of a particular interest when the nominal controller is given in terms of a verbal specification and/or hand-written PLC code.

SUMMARY AND FURTHER READING

This paper provides a concise overview to the synthesis of fault-tolerant discrete-event systems in a language based framework. Individual approaches have been selected to cover active and passive fault tolerant control, as well as post-fault recovery and fault-hiding. Definitions from the original literature have been restated in a homogeneous notation, however, with some simplification and in a less general form. For the precise technical background the reader is kindly referred to the respective literature.

To complement the references provided in this paper, we conclude by accounting for related work, that did not quite fit the framework chosen for this overview. Rohloff (2005) addresses the specific situation of faulty sensors and proposes to represent the effect of a fault be an according variation of the projection operator chosen for observations. The cited reference gives detailed account on modelling and verification, as well as an outline of possible synthesis procedures. Nke and Lunze (2011a,b) discuss fault-tolerant control for automata with inputs and outputs. The contributions include a systematic approach to model sensor and actuator faults as well as a synthesis procedure for reconfiguration to achieve fault tolerance w.r.t. prescribed performance objectives. Sülek and Schmidt (2013) consider faults with the effect, that certain events can no longer occur. The discussion includes a synthesis procedure to achieve fault tolerance in the closed-loop configuration. Moor and Schmidt (2015) address fault-tolerance in a hierarchical control architecture. In such a setting, a lower-bound language

[10] The actual setting in the cited literature is more general in that it explicitly accounts for high-level events and low-level events for component interconnection in a hierarchical control architecture. While Wittmann et al. (2013) applies to prefix-closed languages only, additional conditions for the general case of not prefix-closed languages are developed in Wittmann (2014).

inclusion specification can have priority over an upper bound, since there is the option to pass on undesired behaviour for compensation further up in the hierarchy.

In the opinion of the author, additional insight could be gained by future research to address fault-tolerance for languages of infinite strings, since this is a natural interpretation domain for non-terminating processes.

REFERENCES

Blanke, M., Kinnaert, M., Lunze, J., Staroswiecki, M., and Schröder, J. (2006). *Diagnosis and Fault-Tolerant Control*. Springer.

Brandt, R.D., Garg, V., Kumar, R., Lin, F., Marcus, S.I., and Wonham, W.M. (1990). Formulas for calculating supremal controllable and normal sublanguages. *Systems and Control Letters*, 15, 111–117.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer, second edition.

Cho, H. and Marcus, S.I. (1989). On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation. *Maths. of Control, Signals & Systems*, 2, 47–69.

Hopcroft, J.E. and Ullman, J.D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading.

Kumar, R., Garg, V., and Marcus, S.I. (1992). On supervisory control of sequential behaviors. *IEEE Transactions on Automatic Control*, 37, 1978–1985.

Kumar, R., Garg, V., and Marcus, S. (1993). Language stability and stabilizability of discrete event dynamical systems. *SIAM J. Control and Optimization*, 31, 1294–1320.

Lin, F. and Wonham, W.M. (1988). On observability of discrete-event systems. *Information Sciences*, 44, 173–198.

Moor, T., Baier, C., Yoo, T.S., Lin, F., and Lafortune, S. (2012). On the computation of supremal sublanguages relevant to supervisory control. *Workshop on Discrete Event Systems (WODES)*, 175–180.

Moor, T. and Schmidt, K. (2015). Fault-tolerant control of discrete-event systems with lower-bound specifications. *Workshop on Dependable Control of Discrete Systems (DCDS)*.

Nke, Y. and Lunze, J. (2011a). A fault modeling approach for input/output automata. In *Proceedings of the 18th IFAC World Congress*. Italy.

Nke, Y. and Lunze, J. (2011b). Online control reconfiguration for a faulty manufacturing process. In *3rd International Workshop on Dependable Control of Discrete Systems (DCDS)*. Germany.

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

Paoli, A., Sartini, M., and Lafortune, S. (2008). A fault tolerant architecture for supervisory control of discrete event systems. *Proceedings of the 17th IFAC world congress*, 6542—6547.

Paoli, A., Sartini, M., and Lafortune, S. (2011). Active fault tolerant control of discrete event systems using online diagnostics. *Automatica*, 47(4), 639–649.

Ramadge, P.J. (1989). Some tractable supervisory control problems for discrete-event systems modeled by büchi automata. *IEEE Transactions on Automatic Control*, 34, 10–19.

Ramadge, P.J. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25, 206–230.

Ramadge, P.J. and Wonham, W.M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77, 81–98.

Richter, J.H. (2011). *Reconfigurable control of nonlinear dynamical systems: fault hiding approach*. LNCIS 408. Springer-Verlag.

Rohloff, K.R. (2005). Sensor failure tolerant supervisory control. In *IEEE Proceedings of the 44th International Conference on Decision and Control*, 3493–3498.

Sampath, M., Sengupeta, R., , Lafortune, S., and Sinnamohideen, K. (1995). Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40:9, 1555–1575.

Schmidt, K. and Breindl, C. (2014). A framework for state attraction of discrete event systems under partial observation. *Information Sciences*, 281, 265–280.

Sülek, A.N. and Schmidt, K.W. (2013). Computation of fault-tolerant supervisors for discrete event systems. In *4th IFAC Workshop on Dependable Control of Discrete Systems*, 115–120.

Sülek, A.N. and Schmidt, K.W. (2014). Computation of supervisors for fault-recovery and repair for discrete event systems. In *Workshop on Discrete Event Systems*, 428–438.

Thistle, J.G. and Lamouchi, H.M. (2009). Effective control synthesis for partially observed discrete-event systems. *SIAM J. Control and Optimization*, 48, 1858–1887.

Thistle, J.G. and Wonham, W.M. (1994). Supervision of infinite behavior of discrete event systems. *SIAM J. Control and Optimization*, 32, 1098–1113.

Wen, Q., Kumar, R., and Huang, J. (2008a). Synthesis of optimal fault-tolerant supervisor for discrete event systems. *American Control Conference*, 1172–1177.

Wen, Q., Kumar, R., and Huang, J. (2014). Framework for optimal fault-tolerant control synthesis: maximize prefault while minimize post-fault behaviors for discrete event systems. *IEEE Transactions on Systems, Man and Cybernetics: Systems*, 44, 1056–1066.

Wen, Q., Kumar, R., Huang, J., and Liu, H. (2008b). A framework for fault-tolerant control for discrete event systems. *IEEE Transactions on Automatic Control*, 53, 1839–1849.

Willner, Y. and Heymann, M. (1995). Language convergence in controlled discrete-event systems. *Automatic Control, IEEE Transactions on*, 40(4), 616–627.

Wittmann, T. (2014). *Zur Methodik und Anwendung fehlerverdeckender Steuerungsrekonfiguration für eine Klasse ereignisdiskreter Systeme*. Dissertation, Friedrich-Alexander Universität Erlangen-Nürnberg.

Wittmann, T., Richter, J., and Moor, T. (2012). Fault-tolerant control of discrete event systems based on fault-accommodating models. *Preprints of the 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS)*, 854–859.

Wittmann, T., Richter, J., and Moor, T. (2013). Fault-hiding control reconfiguration for a class of discrete-event systems. *Workshop on Dependable Control of Discrete Systems (DCDS)*.

Wonham, W.M. (1999). Notes on control of discrete event systems. Technical report, Department of Electrical & Computer Engineering, University of Toronto.

Yin, X. and Lafortune, S. (2014). A general approach for synthesis of supervisors for partially-observed discrete-event systems. *Proc. 19th IFAC World Congress*, 2422–2428.