Annual Reviews in Control 41 (2016) 159–169

Contents lists available at ScienceDirect

Annual Reviews in Control

journal homepage: www.elsevier.com/locate/arcontrol

A discussion of fault-tolerant supervisory control in terms of formal languages

Thomas Moor

Lehrstuhl für Regelungstechnik, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

ARTICLE INFO

Article history:
Received 29 October 2015
Revised 20 January 2016
Accepted 4 February 2016
Available online 4 May 2016

Keywords:
Discrete-event systems
Supervisory control
Fault-tolerant control
Passive fault-tolerant control
Active fault-tolerant control
Post-fault recovery
Fault-hiding approach

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. The present paper addresses discrete-event systems and provides an overview on fault-tolerant supervisory control. The discussion is held in terms of formal languages to uniformly present approaches to passive fault-tolerance, active fault-tolerance, post-fault recovery and fault hiding.

1. Introduction

A fault is considered a sudden change in the behaviour of a system with potentially undesired consequences. In technical applications, the overall effect of a single faulty component can range from degraded performance up to total breakdown, including environmental damage or human operator injury. By a *fault-tolerant design* one seeks to avoid such negative consequences in a systematic manner and up to a prescribed degree. Here, a common approach is to relate the reliability of individual components and the dependencies among different components with the overall functionality regarding safety and performance; see e.g. Dubrova (2013) for an introduction to fault-tolerant design from this perspective.

When it comes to control, the system consists of a plant and a controller where the latter is interpreted as a degree of freedom in the design of the overall closed-loop behaviour. Assuming that the plant is subject to a fault, one requires the controller to compensate the fault to some degree in order to maintain an operational closed loop with a well defined overall performance that complies to relevant safety requirements. Such a controller is termed *fault tolerant*, with *fault-tolerant control* as a particular approach to fault-tolerant design. For its obvious practical relevance, fault-tolerant control has received extensive attention in the literature; see e.g. Blanke, Kinnaert, Lunze, Staroswiecki, and Schröder (2006) for a comprehensive study.

Given the *nominal plant behaviour* in the absence of the fault and the *degraded plant behaviour* after the occurrence of the fault, the literature proposes two alternative strategies to achieve a fault-tolerant closed-loop system. First, one may design a single controller that can handle both plant models satisfactory. This is referred to as *passive fault-tolerant control* and is closely related to robust control. In contrast to plain robust control, however, attention needs to be paid to the transient behaviour when the fault occurs. Moreover, depending on the system classes under consideration, passive fault-tolerant control may impose unacceptable limitations on the nominal closed-loop behaviour. As a second strategy, one may refer to methods related to adaptive control and design one controller for the nominal plant, one controller for the degraded plant and a diagnoser to detect the fault. The latter is used to activate the appropriate controller. This strategy is referred to as *active fault-tolerant control*. In contrast to the common setting in adaptive control, the particular challenge again is the switching, now with three modes of operation: (a) no fault, (b) fault has occurred but is not yet diagnosed, and (c) fault present and diagnosed. In particular during (b) the degraded plant is under nominal control and may fail to satisfy even elementary requirements like stability.

E-mail address: lrt@fau.de

http://dx.doi.org/10.1016/j.arcontrol.2016.04.001

A comparative study of active and passive fault-tolerant control is given by Jiang and Yu (2012).

As a further variation of active fault-tolerant control, a so-called *reconfiguration block* can be used to adapt the nominal controller to the faulty plant by hiding the effect of the fault. To further illustrate the *fault-hiding approach*, consider the situation of a sensor failure. Here, one can implement an observer with the reconfiguration block that, together with the faulty plant, mimics the nominal plant behaviour. With this approach, the nominal controller remains active even when the fault occurs and, hence, the closed-loop performance may benefit from advanced tuning strategies used for the nominal case. Virtual sensors and likewise virtual actuators are discussed by Blanke et al. (2006), for more general forms of fault-hiding see Steffen (2005) and Richter (2011).

The references provided so far focus attention on continuous-signal plant models represented by ordinary differential equations. In contrast, the present paper discusses the synthesis of fault-tolerant control for discrete-event systems that are adequately representable by regular languages. As a general framework for the control of this system class we refer to supervisory control as proposed by Ramadge and Wonham (1987,1989) and demonstrate in a concise and homogeneous notation how the above strategies for fault tolerance can be applied.

A preliminary observation for the system class under consideration is that passive fault-tolerant control plays a special role: since discrete-event systems model sudden changes of behaviour seamlessly, any switching scheme introduced to achieve fault-tolerance can be alternatively interpreted as passive fault-tolerant control. Moreover, an ordinary event can be used to represent the occurrence of the fault. Thus, a *fault-accommodating model* that summarises the nominal plant behaviour and the degraded plant behaviour still belongs to the same system class as the nominal plant and solutions to the synthesis problem for passive fault-tolerant control can be obtained by the very same established procedures known from the nominal synthesis problem. We refer to this perspective as *naive fault-tolerant control*. In general, we expect that alternative control architectures motivated by additional control objectives or specific design strategies also comply with the naive setting.

In this paper, we discuss approaches to the synthesis of fault-tolerant supervisory control provided by the literature. We make use of a homogeneous notation in order to demonstrate how the approaches relate to the naive approach as a common technical base. Observing applicable constraints and conducting the discussion up to a relevant level of detail, we focus attention to active fault-tolerant control (Paoli, Sartini, & Lafortune, 2008,2011) and fault hiding (Wittmann, 2014; Wittmann, Richter, & Moor, 2013) for specific control architectures, as well as variants of post-fault recovery (Sülek & Schmidt, 2014; Wen, Kumar, & Huang, 2008a, 2014; Wen, Kumar, Huang, & Liu, 2008b) for additional control objectives.

To complement the references further discussed in the body of this paper, we account for related work, that does not fit the language based framework. Park and Lim (1998) propose a notion of fault tolerance in terms of reachability of marked states. The discussion includes a characterisation of the existence of a fault tolerant controller that in addition exhibits a robustness property. Rohloff (2005) addresses the specific situation of faulty sensors and proposes to represent the effect of a fault by an according variation of the projection operator chosen for observations. The cited reference gives detailed account on modelling and provides a procedure to test for fault-tolerance, as well as an outline of possible synthesis procedures. Girault and Rutten (2009) adapt methods from supervisory control for the synthesis of fault-tolerant programs. A particular focus here is on the systematic generation of models for certain classes of faults and for certain classes of components subject to a fault. The cited work uses labelled transitions systems with guards and actions as a modelling framework. Nke and Lunze (2011a,2011b) 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 and discuss the option to pass on undesired post-fault behaviour for compensation further up in the hierarchy.

The paper is organised as follows. A language based framework for the control of discrete-event systems is introduced in Sections 2 and 3, as a variation of *supervisory control under partial observation* originally proposed by Lin and Wonham (1988) and referring to Ramadge and Wonham (1987). As a further development of Wittmann, Richter, and Moor (2012), Section 4 elaborates the naive approach to fault-tolerant control to motivate closed-loop requirements relevant for fault tolerance. The subsequent discussion addresses active fault-tolerant control in Section 5, post-fault recovery in Section 6 and the fault-hiding approach in Section 7. We conclude with a summary. The paper is an extended transcript of a plenary talk held at the *5th International Workshop on Dependable Control of Discrete-Event Systems (5th IFAC DCDS 2015), Mexico*; see Moor (2015) for the corresponding conference contribution.

2. 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 $s = \sigma_1\sigma_2\ldots\sigma_n, n \in \mathbb{N}, \sigma_i \in \Sigma$, 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 $|\epsilon| = 0$. Given two strings $s = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma^*$ and $t = \tau_1\tau_2\ldots\tau_m \in \Sigma^*$, the *concatenation* is defined $st := \sigma_1\sigma_2\ldots\sigma_n\tau_1\tau_2\ldots\tau_m \in \Sigma^*$ with $s\epsilon = s = \epsilon s$. 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., $s = (\mathrm{pre}_n s)(\mathrm{suf}_n s)$.

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 *prefix-closed* (or short *closed*) if $L = \mathrm{pre}\, L$. A language K is *relatively prefix-closed w.r.t. L* (or short *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 $LM := \{st \mid s \in L,\ t \in M\}$. The concatenation of closed languages is

closed. The *relative suffix* or *quotient* is defined $L/M := \{ t \mid \exists s \in M : st \in L \}$. Note that $(\mathrm{pre}\, L)/M = \mathrm{pre}\,(L/M)$ and, 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 $M \Leftarrow K$, if there is a non-negative integer n such that for each $s \in K$, there exists $i \leq n$ 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 is equivalent to the existence of a non-negative integer n such that $K \subseteq (\cup_{i \leq n} \Sigma^i) M$. The latter inclusion is also proposed by Kumar, Garg, and Marcus (1993) to define the notion of *language stability*. Given three languages $K, M, N \subseteq \Sigma^*$, K is said to *converge finitely to M after N* if $M \Leftarrow K/N$.

For the *observable events* $\Sigma_o \subseteq \Sigma$, the *natural projection* $\mathrm{p_o} : \Sigma^* \to \Sigma_o^*$ 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 as well as language concatenation commute 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 $(\Sigma_1 \cup \Sigma_2)^*$ to Σ_1^* and Σ_2^*, respectively. Here, L_1 and L_2 are said to be *non-conflicting*, if $\mathrm{pre}\,(L_1 \parallel L_2) = (\mathrm{pre}\, L_1) \parallel (\mathrm{pre}\, L_2)$. 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*. In this case, L_1 and L_2 are non-conflicting.

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 $(\mathrm{pre}\, K)\Sigma_{uc} \cap (\mathrm{pre}\, L) \subseteq \mathrm{pre}\, K$. 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. 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.

3. 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.

3.1. Modelling

Consider a processes that, by assumption, 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 process, individual transitions are labelled with *events* from a finite alphabet Σ to be externally visible. We restrict attention to processes 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*. Then, an observation of the process for some arbitrarily long but finite physical duration 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 observations is regarded a *discrete-event system* that models the physical process under consideration. Note that, by definition, $s \in L$ implies $\mathrm{pre}\, s \subseteq L$, i.e., the models considered so far are prefix-closed languages. We emphasise this fact by denoting the model $\mathrm{pre}\, L$, also referred to as the *local behaviour*.

3.2. Elementary properties

Informally, a process exhibits a *safety property* if "something bad cannot happen". In the proposed modelling framework this can be expressed as a set inclusion

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

with $E \subseteq \Sigma^*$ the complement of the "bad strings". Since the local behaviour on the left-hand side of the inclusion is prefix-closed, E can be substituted by its supremal prefix-closed sublanguage without affecting the imposed constraint. Therefore any safety property can be represented by a closed upper bound on the local behaviour.

In contrast to safety, a *liveness property* requires that "something good will happen"; see Manna and Pnueli (1990), Baier and Kwiatkowska (2000) for a detailed classification. For the purpose of the present paper, we recall two liveness properties commonly discussed in the context of $*$-languages. A local behaviour $\mathrm{pre}\, L \subseteq \Sigma^*$ *does not deadlock*, if any generated event sequence can be extended by one more event i.e., if L is *complete*

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

In order to obtain a liveness property in the intended sense, the formal requirement Eq. (2) imposed on the model needs to be accompanied by an additional assumption regarding the process itself: *if, at any physical time, the process can generate one more event then the process will generate one more event*. Then, a non-empty local behaviour that does not deadlock models a process that within infinite physical time generates an infinite number of events. Such processes are also referred to as *non-terminating processes*.

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

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

i.e., if there is the persistent possibility to attain an accepted string. For 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 process has the chance to generate an accepted string, then it will eventually do so*. A process that does not livelock w.r.t. an accepted behaviour can be represented as a single language $L \subseteq \Sigma^*$ with associated local behaviour $\mathrm{pre}\, L$ and associated accepted behaviour L. This corresponds to the generated language and the marked language of a non-blocking automaton realisation, respectively. 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.

Remark 1. For processes that do deadlock or livelock, 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 process 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

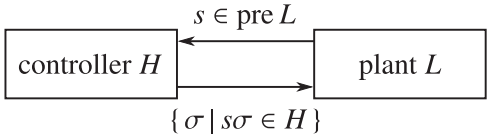

Fig. 1. Closed-loop configuration.

place to require that the controller implicitly prevents any occurrence of the blocking event.

3.3. System composition

When a process is composed from multiple components, one seeks to systematically construct an overall model from individual component 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 local behaviour of each individual component. Thus, when considering local behaviours $\operatorname{pre} L_1 \subseteq \Sigma_1^*$ and $\operatorname{pre} L_2 \subseteq \Sigma_2^*$, the parallel composition

$$L_{\mathrm{loc}} := (\operatorname{pre} L_1) \parallel (\operatorname{pre} L_2) \tag{4}$$

is an adequate representation of the local behaviour of the overall process. Applying the same formula to the accepted behaviours $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$,

$$L := L_1 \parallel L_2, \tag{5}$$

the composition amounts to the requirement that accepted configurations are attained simultaneously by both components. If both accepted behaviours are non-conflicting, we obtain $\operatorname{pre} L = L_{\mathrm{loc}}$ and, hence, the discrete-event system L is again an adequate model of the composition. If, on the other hand, the accepted behaviours fail to be non-conflicting, we appeal to Remark 1 and propose to merge accepted and local behaviour by an explicit blocking event. For non-terminating processes, a common alternative to simultaneous acceptance is to require that an infinite string is generated with infinitely many prefixes accepted for either one component.

3.4. Closed-loop configuration

For the purpose of control, the alphabet is composed as a disjoint union of *controllable events* and *uncontrollable events*. To this end, we consider the *plant* to be given as a discrete-event system and we assume that the underlying process is equipped with some interface to disable any controllable event at any time. A supervisor is then defined as causal feedback that maps the past event sequence to a so called *control-pattern* and thereby indicates which events are enabled. This basic setting is extended to account for the situation of *partial observation* by distinguishing *observable events* and *unobservable events*. Then, the supervisor shall apply consistent control patterns after the generation of event sequences that cannot be distinguished by observation.

In this paper, we restrict attention to the special case where the controller cannot disable unobservable events and represent the causal feedback map as a language H to interpret supervision as a form of system composition. Technically, $\{\sigma \mid s\sigma \in H\} \subseteq \Sigma$ corresponds to the control pattern applied after the generation of $s \in \operatorname{pre} L$ from the local plant behaviour; see Fig. 1. To parallel the setting of *non-blocking supervisory control under partial observation* in Lin and Wonham (1988), we impose the following technical conditions on the controller H.

Definition 2. Given an alphabet with the common partition $\Sigma = \Sigma_c \dot\cup \Sigma_{uc} = \Sigma_o \dot\cup \Sigma_{uo}$ in controllable, uncontrollable, observable and unobservable events, respectively, a language $H \subseteq \Sigma^*$ is an *admissible controller* for the plant $L \subseteq \Sigma^*$, if

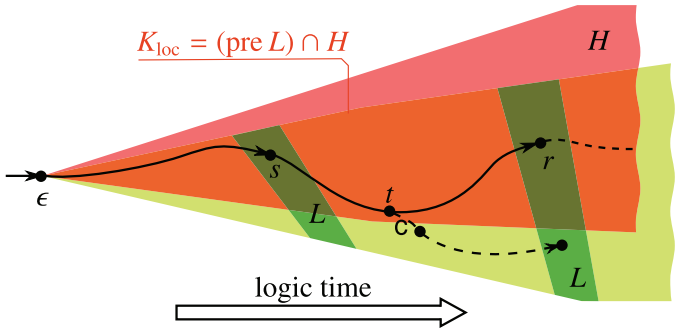

Fig. 2. Closed-loop behaviour as language intersection.

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

In the above setting, $K_{\mathrm{loc}} := (\operatorname{pre} L) \parallel (\operatorname{pre} H)$ represents the local closed-loop behaviour and, by [H0] and [H2], amounts to $K_{\mathrm{loc}} = (\operatorname{pre} L) \cap H$. This intersection is illustrated in Fig. 2, with $\operatorname{pre} L$ as the light green region, H the light red region, and the intersection K_{loc} appearing in orange by overlay. The generation of an event sequence always starts with the empty string ϵ for successive concatenation of further events with the progress of logic time, i.e., a point in the diagram represents the event sequence generated so far. Event sequences generated by the plant remain in $\operatorname{pre} L$ (light green) and eventually attain an accepted string in L (solid green). The example sequence $s \in L$ happens to be also compliant with the controller H, i.e., we have $s \in K_{\mathrm{loc}}$. Continuing on the indicated sequence up to $t \in K_{\mathrm{loc}}$, $s < t$, the illustration renders the event $\mathrm{c} \in \Sigma$ as a possible successor that is compliant to the plant but disabled by the controller. To satisfy the controllability requirement [H1], c must be a controllable event.

In analogy with the local closed-loop behaviour K_{loc} we obtain $K := L \parallel H = L \cap H$ as the accepted closed-loop behaviour. Here, condition [H4] requires that any string compliant with the local closed loop K_{loc} must allow for an extension that continues to be compliant with K_{loc} while becoming accepted by the plant. Inspecting again the example given by Fig. 2, disabling the event c as an immediate successor of t is consistent with [H4] since $r \in L$, $t < r$, still demonstrates the existence of the required extension for t. Technically, we obtain $K_{\mathrm{loc}} = \operatorname{pre} K$, i.e., the local closed-loop behaviour does not livelock w.r.t. K and K is a discrete-event system that models the synchronous composition of plant and controller. The completeness requirement [H3] then requires that K represents a non-terminating process. Referring to the example Fig. 2, [H3] requires the existence of a proper extension of r that complies with K_{loc}.

The following theorem relates the slightly different setting used in the present paper to results by Lin and Wonham (1988), and establishes a characterisation of achievable closed-loop behaviours.

Theorem 3. *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 $K = L \cap H$.

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 $\mathrm{pre}\, K = (\mathrm{pre}\, L) \cap (\mathrm{pre}\, H)$. 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 $H := \mathrm{p_o}^{-1}\, \mathrm{p_o}((\mathrm{pre}\, K)\, \Sigma_{\mathrm{uc}}^*)$, which is observed to satisfy [H0]–[H2] by elementary properties of the relevant operators. The crucial step is to refer to [K1] and [K2] in order to obtain $(\mathrm{pre}\, L) \cap (\mathrm{pre}\, H) = \mathrm{pre}\, K$. Then, [K0] implies $L \cap H = K$. The last two equations are used to obtain [H3] from [K3] and, finally, [H4]. □

3.5. 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 $K = L \cap H \subseteq E$. A core observation from the literature is that the closed-loop properties [K0]–[K3] are retained under arbitrary union; e.g. Ramadge and Wonham (1989) for controllability, Lin and Wonham (1988) for normality, and Kumar, Garg, and Marcus (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 $H_{\mathrm{n}}^\uparrow \neq \emptyset$ 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 and for specific combinations of closed-loop properties, various such procedures have been proposed; see e.g. Cho and Marcus (1989), Brandt et al. (1990), Kumar et al. (1992) as well as Moor, Baier, Yoo, Lin, and Lafortune (2012) for the particular situation of the present paper.

4. Naive fault-tolerant control

Compliant with Blanke et al. (2006), a fault is considered a sudden change in the behaviour of the plant with potentially negative consequences for the overall performance. A particular feature of the class of discrete-event systems under consideration is to seamlessly model such sudden changes. Referring to the introduction of the present paper, a possible strategy to achieve a fault-tolerant design is to first construct an overall model that accommodates for the fault and then to apply established methods for controller synthesis as presented in the previous section. We are now in the position to further elaborate this *naive approach to fault-tolerant control* proposed by Wittmann et al. (2012).

We begin with a nominal closed-loop configuration, consisting of a nominal alphabet denoted Σ_{n} with the common partitioning, a nominal plant model $L_{\mathrm{n}} \subseteq \Sigma_{\mathrm{n}}^*$, a nominal language-inclusion specification $E_{\mathrm{n}} \subseteq L_{\mathrm{n}}$, and an admissible nominal controller $H_{\mathrm{n}} \subseteq \Sigma_{\mathrm{n}}^*$ according to the requirements [H0]–[H4] with resulting closed-loop behaviour $K_{\mathrm{n}} := L_{\mathrm{n}} \cap H_{\mathrm{n}} \subseteq E_{\mathrm{n}}$. To accommodate for the fault, the alphabet is extended by a distinguished event $\mathrm{f} \notin \Sigma_{\mathrm{n}}$, $\Sigma_{\mathrm{f}} := \Sigma_{\mathrm{n}} \dot\cup \{\mathrm{f}\}$, and we define the *degraded plant behaviour* $L_{\mathrm{d}} \subseteq \Sigma_{\mathrm{f}}^*$ to specify all possible pasts that may trigger the fault and the corresponding post-fault behaviour. In particular, we may assume that

$$L_{\mathrm{d}} \subseteq (\mathrm{pre}\, L_{\mathrm{n}})\mathrm{f}\Sigma_{\mathrm{f}}^*\,, \tag{6}$$

with the disjoint union

$$L_{\mathrm{f}} = L_{\mathrm{n}} \dot\cup L_{\mathrm{d}} \tag{7}$$

as the overall *fault-accommodating model*. This construct is illustrated in Fig. 3. There, the nominal model L_{n} and the degraded model L_{d} are given in solid green and solid red, respectively. The prefixes are given in the corresponding light colour, with the intersection in orange. The process starts with the empty string ϵ to generate events that assemble a monotone sequence within the prefix of the fault-accommodating model and to eventually attain an accepted configuration. Since $\epsilon \in (\mathrm{pre}\, L_{\mathrm{n}}) \cap (\mathrm{pre}\, L_{\mathrm{d}})$, the sequence initially evolves within this intersection. If no fault occurs, the generated sequence remains within the local nominal behaviour and eventually attains a configuration accepted by the nominal model; see $t \in L_{\mathrm{n}}$ in Fig. 3. If the fault occurs, the sequence leaves the intersection by the fault event f to continue within the local degraded behaviour $\mathrm{pre}\, L_{\mathrm{d}}$ and to eventually attain a configuration accepted by the degraded model; see $s \in L_{\mathrm{d}}$ in Fig. 3.

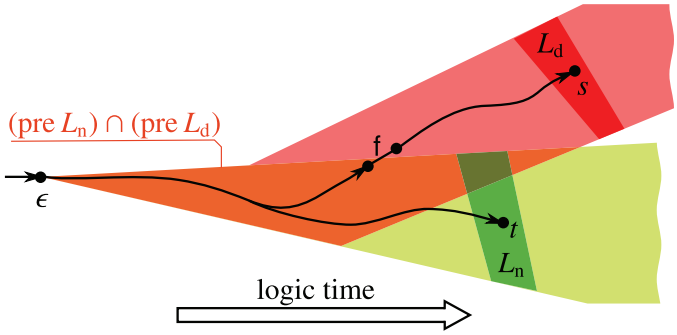

Fig. 3. Fault-accommodating model as a union composition.

The following proposition states immediate consequences of Assumption (6).

Proposition 4. *With $\Sigma_{\mathrm{f}} = \Sigma_{\mathrm{n}} \dot\cup \{\mathrm{f}\}$, consider two languages $L_{\mathrm{n}} \subseteq \Sigma_{\mathrm{n}}^*$ and $L_{\mathrm{d}} \subseteq \Sigma_{\mathrm{f}}^*$ compliant with Assumption (6). Then, $L_{\mathrm{f}} = L_{\mathrm{n}} \cup L_{\mathrm{d}}$ satisfies:*

$$L_{\mathrm{d}} \cap \Sigma_{\mathrm{n}}^* = \emptyset\,, \tag{8}$$

$$(\mathrm{pre}\, L_{\mathrm{d}}) \cap \Sigma_{\mathrm{n}}^* \subseteq \mathrm{pre}\, L_{\mathrm{n}}\,, \tag{9}$$

$$L_{\mathrm{f}} \cap \Sigma_{\mathrm{n}}^* = L_{\mathrm{n}}\,, \tag{10}$$

$$(\mathrm{pre}\, L_{\mathrm{f}}) \cap \Sigma_{\mathrm{n}}^* = \mathrm{pre}\, L_{\mathrm{n}}\,. \tag{11}$$

Proof-Outline. The claim follows by elementary properties of the relevant operators; see also Section 2. □

As a consequence of Eqs. (10) and (11), we obtain

$$(\mathrm{pre}\, L_{\mathrm{f}}) \cap \Sigma_{\mathrm{n}}^* = \mathrm{pre}\, (L_{\mathrm{f}} \cap \Sigma_{\mathrm{n}}^*)\,, \tag{12}$$

i.e., L_{f} and Σ_{n}^* are non-conflicting. In other words: liveness properties encoded in the fault-accommodating model by construction are consistent with the hypothesis that the fault may not occur at all. This observation suggests some further considerations. Technically, the local behaviour of a discrete-event system may account for event sequences that imply the occurrence of a particular event, either as an immediate successor or in attaining an accepted string. Regarding the fault, however, it is a sensible assumption that its occurrence under no circumstances becomes an inevitable consequence of the event sequence generated so far. This is expressed by the following conditions imposed in the fault-accommodating model:

$$(\forall s)(\exists t \in \Sigma_{\mathrm{n}}^*)[\, s \in \mathrm{pre}\, L_{\mathrm{f}} \Rightarrow st \in L_{\mathrm{f}}\,]\,, \tag{13}$$

$$(\forall s)(\exists \sigma \in \Sigma_{\mathrm{n}})[\, s\mathrm{f} \in \mathrm{pre}\, L_{\mathrm{f}} \Rightarrow s\sigma \in \mathrm{pre}\, L_{\mathrm{f}}\,]\,. \tag{14}$$

A *fault-accommodating language-inclusion specification* can be set up following the same pattern as for the plant model; i.e., we parametrise an upper bound by the union composition

$$E_{\mathrm{f}} = E_{\mathrm{n}} \dot\cup E_{\mathrm{d}}\,, \tag{15}$$

subject to the assumption $\mathrm{pre}\, E_d \subseteq (\mathrm{pre}\, E_n)\mathrm{f}\Sigma_f^*$ obtained by uniform substitution in Eq. (6), and with consequences as given by Proposition 4. In particular, we have $E_f \cap \Sigma_n^* = E_n$, i.e., up to the occurrence of the fault the fault-accommodating specification matches the nominal specification. Here, the intention of E_d is to relax E_n after the occurrence of the fault.

Once fault-accommodating models of plant and specification are provided, options are to test whether an existing controller (e.g. the nominal controller) is fault tolerant or to synthesise a fault-tolerant controller from scratch. Both problems can be addressed by the same procedures as used for the nominal control problem, but now applied to the fault-accommodating models as input data with the fault event regarded as uncontrollable. In general, the fault event is also regarded unobservable, however, depending on the level of abstraction one also encounters applications where the plant instantly reports the fault by built-in diagnosis.

Depending on the particular application at hand, an adaption of the desired closed-loop properties [K0]–[K3] to the interpretation of the fault event regarding liveness properties may be required. In analogy to Conditions (13) and (14), one may ask the controller not to provoke the fault by disabling all alternative events in the closed-loop system, i.e., one may impose the additional closed-loop requirements

[K4] $(\forall s)(\exists t \in \Sigma_n^*)[\, s \in \mathrm{pre}\, K_f \Rightarrow st \in K_f\,]$,
[K5] $(\forall s)(\exists \sigma \in \Sigma_n)[\, s\mathrm{f} \in \mathrm{pre}\, K_f \Rightarrow s\sigma \in \mathrm{pre}\, K_f\,]$,

with $K_f := L_f \cap H_f$ the closed-loop behaviour formed by the fault accommodating plant L_f and an admissible controller H_f. As with [K0]–[K3], the closed-loop properties [K4] and [K5] are retained under arbitrary union.

Proposition 5. *Consider an alphabet $\Sigma_f = \Sigma_n \dot\cup \{\mathrm{f}\}$ and a family of languages $(K_a)_{a\in A}$, $K_a \subseteq \Sigma_f^*$ with union $K := \cup_{a\in A} K_a$. If, for all $a \in A$, K_a possesses [K4] or [K5], then K exhibits [K4] and [K5], respectively.*

Proof. To establish [K4] or [K5] for K, pick s or $s\mathrm{f}$ in $\mathrm{pre}\, K$, respectively. Since the prefix operator commutes with arbitrary unions, we can pick $a \in A$ such that $s \in \mathrm{pre}\, K_a$ or $s\mathrm{f} \in \mathrm{pre}\, K_a$, respectively. Referring to the respective property for K_a, we obtain the existence of $t \in \Sigma_n^*$ or $\sigma \in \Sigma_n$, such that $st \in K_a \subseteq K$ or $s\sigma \in \mathrm{pre}\, K_a \subseteq \mathrm{pre}\, K$, respectively. □

Thus, given the fault-accommodating model and a language-inclusion specification, there exists a unique supremal closed-loop behaviour that possesses the properties [K0]–[K5] and that satisfies the specification. For regular parameters a synthesis procedure can be obtained by the framework presented in Moor et al. (2012), with a separate discussion regarding [K4]. Referring to Theorem 3 and [K0]–[K3], one then extracts an admissible controller to implement the supremal achievable closed-loop behaviour. In particular, the resulting controller enforces the specification and the additional closed-loop requirements [K4] and [K5]. By the following theorem the latter two properties ensure that the controller remains admissible under the optional hypothesis, that the fault does not occur at all.

Theorem 6. *Consider an alphabet with the common partition $\Sigma_f = \Sigma_n \dot\cup \{\mathrm{f}\} = \Sigma_c \dot\cup \Sigma_{uc} = \Sigma_o \dot\cup \Sigma_{uo}$ and a plant $L_f \subseteq \Sigma_f^*$. If a controller H_f is admissible to L_f and if the closed loop $K_f = L_f \cap H_f$ satisfies [K0]–[K5], then H_f is also admissible to $L_f \cap \Sigma_n^*$.*

Proof. Inspecting Definition 2, admissibility of H_f to L_f and to $L_f \cap \Sigma_n^*$ is equivalent to admissibility of H_f to L_f and the following two additional properties:

[H5] $(\mathrm{pre}\, L_f) \cap (\mathrm{pre}\, H_f) \cap \Sigma_n^*$ is complete, and
[H6] $L_f \cap \Sigma_n^*$ and H_f are non-conflicting.

As a second preliminary observation, note that [K4] implies that K_f and Σ_n^* are non-conflicting, and, that [K5] together with completeness [K3] implies that $(\mathrm{pre}\, K_f) \cap \Sigma_n^*$ is complete. Regarding [H5], we obtain with [H4] that $(\mathrm{pre}\, L_f) \cap (\mathrm{pre}\, H_f) \cap \Sigma_n^* = (\mathrm{pre}\, K_f) \cap \Sigma_n^*$, which is complete. Regarding [H6], we obtain with [H4] that $(\mathrm{pre}\,(L_f \cap \Sigma_n^*)) \cap (\mathrm{pre}\, H_f) \subseteq (\mathrm{pre}\, L_f) \cap (\mathrm{pre}\, H_f) \cap \Sigma_n^* = (\mathrm{pre}\,(L_f \cap H_f)) \cap \Sigma_n^* = (\mathrm{pre}\, K_f) \cap \Sigma_n^* = \mathrm{pre}\,(K_f \cap \Sigma_n^*) = \mathrm{pre}\,(L_f \cap \Sigma_n^* \cap H_f)$, to establish non-conflictingness. □

To compare the resulting fault-tolerant control with the nominal case, consider controllers H_f and H_n obtained for the respective input data. Observe that the above theorem implies admissibility of $H_f \cap \Sigma_n^*$ to $L_f \cap \Sigma_n^*$. With Assumption (6) in place for both the fault-accommodating plant and the fault-accommodating specification, and referring to Proposition 4, we obtain admissibility of $H_f \cap \Sigma_n^*$ to L_n and $L_n \cap (H_f \cap \Sigma_n^*) \subseteq E_n$. In particular, $H_f \cap \Sigma_n^*$ solves the nominal control problem. Thus, assuming H_n minimally restrictive, we obtain

$$L_f \cap H_f \cap \Sigma_n^* \subseteq L_n \cap H_n\,. \tag{16}$$

However, even if we assume H_f minimally restrictive, we cannot expect equality in the above inclusion. This is because a fault-accommodating plant implies that a fault-tolerant controller avoids those pre-fault configurations, from which, in the case of the fault, the post-fault requirements cannot 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 must not be considered a fundamental limitation of the presented naive approach to fault-tolerant control, but a consequence of the respective input data at hand. Moor and Schmidt (2015) address this situation by proposing a systematic relaxation of the fault-accommodating specification in order to achieve equality in Eq. (16).

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. On the other hand, and provided that one achieves equality in (16), an interpretation as *active* fault-tolerant control is obtained by considering any deviation of the fault-tolerant controller from the behaviour of the nominal controller as a post-fault switching of controllers.

We conclude this section with a discussion of *persistent faults*, i.e., faults that can only occur once, technically characterised by

$$K_f \subseteq L_f \subseteq \Sigma_n^*\{\epsilon, \mathrm{f}\}\Sigma_n^*\,. \tag{17}$$

Then, [K4] and [K5] can be equivalently stated as completeness and non-conflictingness properties.

Proposition 7. *For a complete closed-loop model $K_f \subseteq \Sigma_n^*\{\epsilon, \mathrm{f}\}\Sigma_n^*$, [K4] and [K5] are satisfied if and only if*

[K4'] *K_f and Σ_n^* are non-conflicting, and*
[K5'] *$(\mathrm{pre}\, K_f) \cap \Sigma_n^*$ is complete.*

Proof-Outline. Each of the four individual implications can be established by elementary transformation of the respective condition. □

The above characterisation allows for the following interpretation of fault-tolerant control in the context of robust control; see e.g. Cury and Krogh (1999) and Bourdon, Lawford, and Wonham (2005).

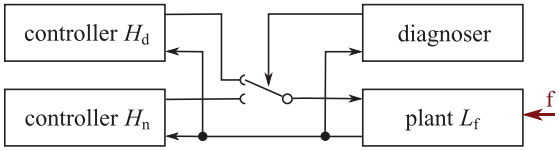

Fig. 4. Active fault-tolerant control with explicit diagnosis.

Theorem 8. *Consider an alphabet with the common partition* $\Sigma_f = \Sigma_n \dot\cup \{f\} = \Sigma_c \dot\cup \Sigma_{uc} = \Sigma_o \dot\cup \Sigma_{uo}$ *and a plant* $L_f \subseteq \Sigma_n^*\{\epsilon, f\}\Sigma_n^*$ *that does not conflict with* Σ_n^*. *Then, if and only if a candidate closed-loop behaviour* K_f *satisfies conditions [K0]–[K5], there exists a controller* H_f *with* $K_f = L_f \cap H_f$ *that is admissible to both* L_f *and* $L_f \cap \Sigma_n^*$.

Proof. First assume that K_f satisfies [K0]–[K5]. Referring to Theorem 3 and [K0]–[K3], there exists a controller H_f admissible to L_f with $K_f = L_f \cap H_f$. Admissibility of the same controller to $L_f \cap \Sigma_n^*$ is then a consequence of Theorem 6. For the converse implication, choose H_f with $K_f = L_f \cap H_f$ and assume admissibility to both L_f and $L_f \cap \Sigma_n^*$. Referring to Theorem 3 we obtain [K0]–[K3]. Regarding [K4] and [K5], we refer to the characterisation [K4'] and [K5'] provided by Proposition 7 and to [H5] and [H6] as consequences of admissibility to $L_f \cap \Sigma_n^*$. For [K4'], we refer to non-conflictingness of L_f and Σ_n^* together with [H6] to obtain $(\mathrm{pre}\, K_f) \cap \Sigma_n^* = (\mathrm{pre}\, L_f) \cap (\mathrm{pre}\, H_f) \cap \Sigma_n^* = (\mathrm{pre}\,(L_f \cap \Sigma_n^*)) \cap (\mathrm{pre}\, H_f) = \mathrm{pre}\,(L_f \cap \Sigma_n^* \cap H_f) = \mathrm{pre}\,(K_f \cap \Sigma_n^*)$. For [K5'], we refer to the first of the above equalities obtain completeness by [H5]. □

5. 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; see Fig. 4. 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) and Paoli et al. (2008,2011) and resemble a simplified variant for interpretation in the context of the naive approach, Section 4.

The discussion is organised in three stages. At the first stage (A), a fault-accommodating plant model with persistent fault is operated under nominal control. In the reading of the present paper, we represent the plant by $L_f \subseteq \Sigma_n^*\{\epsilon, f\}\Sigma_n^*$ obtained by the union construction from the previous section subject to Assumption (6), and, in particular, with $L_n = L_f \cap \Sigma_n^*$ as the associated nominal behaviour. To form a closed loop with the nominal controller, the latter is formally interpreted w.r.t. the full alphabet Σ_f. In the subsequent discussion, we refer to this construction by denoting $H_n \subseteq \Sigma_f^*$ the extended nominal controller with $H_n \cap \Sigma_n^*$ the original nominal controller. Technically, the construction amounts to the additional property $H_n = \mathrm{p}_n^{-1}\,\mathrm{p}_n\, H_n$ with the natural projection $\mathrm{p}_n : \Sigma_f^* \to \Sigma_n^*$. In particular, $H_n \cap \Sigma_n^*$ is assumed to be admissible to L_n. This implies that H_n itself satisfies [H0]–[H2]. For a concise notation, the local closed-loop behaviour $K_{loc} = (\mathrm{pre}\, L_f) \cap H_n$ is assumed not to deadlock, which amounts to [H3]. This assumption is not restrictive: if K_{loc} does deadlock, this can be accounted for by extending locking strings by a distinguished event and by regarding this event as a *forbidden string* in the subsequent discussion of safe diagnosability.

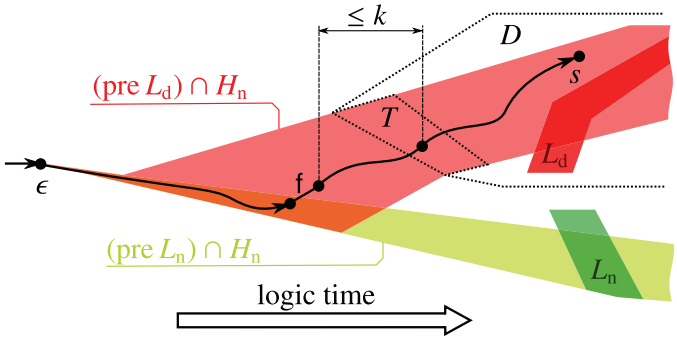

Fig. 5. Diagnosis conditions D and T.

The authors require a diagnoser to report any fault before the system violates a prescribed post-fault safety specification, parametrised as a set of forbidden substrings $\Phi \subseteq \Sigma_f^*$. More precisely, the post-fault specification amounts to the upper bound

$$E_{phi} := \Sigma_f^* - \Sigma_n^* f \Sigma_f^* \Phi \Sigma_f^* \tag{18}$$

to which the local behaviour K_{loc} must comply until the fault is detected. 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, Sengupeta, Lafortune, and Sinnamohideen (1995); for an overview on discrete-event systems diagnosis see also Zaytoon and Lafortune (2013).

Definition 9. Consider a local closed-loop behaviour $K_{loc} = (\mathrm{pre}\, L_f) \cap H_n$ that does not deadlock. With the *diagnosis condition* D defined by

$$D := \{s \in \Sigma_f^* \mid K_{loc} \cap (\mathrm{p}_o^{-1}\,\mathrm{p}_o\, s) \subseteq \Sigma_n^* f \Sigma_f^*\}, \tag{19}$$

K_{loc} is *diagnosable* if there exists a non-negative integer k such that

$$K_{loc} \cap (\Sigma_n^* f \Sigma_f^k) \subseteq D. \tag{20}$$

If in addition

$$T := \{s \in K_{loc} \mid (\mathrm{pre}\, s) \cap D = s\} \subseteq E_{phi}, \tag{21}$$

then K_{loc} is *safe diagnosable*.

The above conditions are illustrated by Fig. 5. There, the local behaviour K_{loc} is given as a nominal component $(\mathrm{pre}\, L_n) \cap H_n$ (light green) and a degraded component $(\mathrm{pre}\, L_d) \cap H_n$ (light red) with the intersection in orange. If no fault occurs, the process generates an event sequence that evolves within $(\mathrm{pre}\, L_n) \cap H_n$ to eventually attain an accepted configuration within the nominal accepted behaviour L_n. The latter is always possible because the nominal controller and the nominal plant are by design non-conflicting. If a fault occurs, the sequence enters $(\mathrm{pre}\, L_d) \cap H_n$ (light red) by the fault event f. Diagnosability Eq. (20) then guarantees that a uniformly bounded amount of logic time after the fault the sequence must enter D. By construction Eq. (19), the *diagnosis condition* $s \in D$ is true if and only if every string that complies to the observation $\mathrm{p}_o\, s$ and the local behaviour K_{loc} includes the fault f. Thus, with the first generation of a string in $K_{loc} \cap D$, the past occurrence of the fault becomes unambiguous. The set T, defined by Eq. (21), consists of precisely those strings that first enter D. In particular, the inclusion in (21) required for safe diagnosability guarantees that the post-fault behaviour does not contain a forbidden substring before the fault becomes unambiguous.

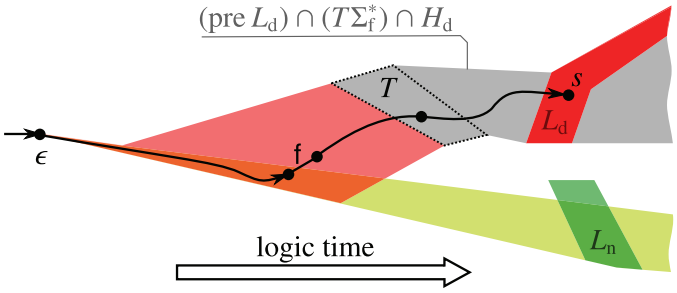

Fig. 6. Effect of post-fault-detection controller.

Note that diagnosability is stated in terms of the local behaviour which cannot incorporate liveness properties other than not to deadlock. This technically justifies the requirement of a uniform upper bound until the fault must be detected. As with language convergence discussed in the following section, the uniform bound in diagnosis may turn out restrictive in certain applications, e.g., for plants that are composed from independent components that possess independent liveness properties. For such situations, a notion of *eventual fault detection* that drops the bound by referring to the associated ω-languages could turn out more appropriate. However, the author of the present paper is not aware of literature in this regard.

At the second stage (B), addressing the time after the fault has been diagnosed, the closed loop K_{loc} shall continue to comply with E_{phi}. In contrast to the situation before the fault was diagnosed, stage (B) allows for a further restriction K_{loc} by control for this particular purpose. The existence of an appropriate control scheme 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.

For the final stage (C), post-fault-detection controllers are synthesised to take over the plant after the diagnosis of the fault, i.e., after the generation of a string in T. The synthesis of these controllers is performed on a strategically constructed formal plant model such 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. The original literature includes a general discussion of design stage (C) to address common control objectives regarding controllability, safety, liveness, and observability. In the reading of the present paper and, in particular, under the assumption that $\Sigma_c \subseteq \Sigma_o$, one may accumulate the post-fault detection controllers to a formal language $H_d \subseteq \Sigma_f^*$ that is capable to take over the plant from H_n once the local behaviour generates a string from T. The composition of the post-fault-detection behaviour $L_f \cap (T\Sigma_f^*)$ with the controller H_d must satisfy the common admissibility criteria from supervisory control. In addition, H_d shall enforce some application specific upper-bound specification $E_d \subseteq \Sigma_f^*$. Here we may assume that E_d does not impose any restrictions until the fault has been detected and that E_d implies the avoidance of forbidden strings after the fault, i.e.,

$$(\mathrm{pre}\, T) \cap (\Sigma_n^* \mathrm{f} \Sigma_f^*) \;\subseteq\; E_d \subseteq E_{\mathrm{phi}}\,. \tag{22}$$

Referring back to Fig. 5, the effect of the post-fault-detection controller is indicated by the grey region in Fig. 6. The figure indicates the avoidance of livelocks as well as the capability of H_d to take over the plant no matter which particular sequence from T reveals the fault. In summary, we impose the following conditions on the post-fault-detection controller:

[A1] H_d is admissible to $L_f \cap (T\Sigma_f^*)$,
[A2] $L_f \cap (T\Sigma_f^*) \cap H_d \subseteq E_d$, and
[A3] $T \subseteq (\mathrm{pre}\, L_f) \cap (T\Sigma_f^*) \cap H_d$.

A minimally restrictive controller $H_d^{\uparrow}$ that complies with [A1] and [A2] can be synthesised by the procedures used for a nominal design applied to the input data $L_f \cap (T\Sigma_f^*)$ and E_d, with a subsequent test regarding [A3]. If the test fails, one concludes that E_d implies a restriction of the local plant behaviour before the detection of the fault and one may consider to relax E_d. If the test of [A3] fails even for $E_d = E_{\mathrm{phi}}$, one concludes that either one of the safe controllability or safe diagnosability conditions are violated.

In order to interpret the proposed scheme of active fault-tolerant control in the context of the naive approach from Section 4, we compose an overall fault-tolerant controller H_f to mimic the switching from H_n to H_d after detection of the fault. Technically, we use

$$C := (D \cap H_n \cap H_d)\Sigma_f^* \tag{23}$$

as the switching condition and define

$$H_f := \{\epsilon\} \cup \{s\sigma \in H_n \,|\, s \notin C\} \cup \{s\sigma \in H_d \,|\, s \in C\}\,. \tag{24}$$

Since $D \cap \Sigma_n^* = \emptyset$, we have $H_f \cap \Sigma_n^* = H_n$ and, hence, equality in (16), as a specific feature of active fault-tolerant control. By the following theorem, the overall controller is consistent with the considerations from the previous Section 4.

Theorem 10. *Consider an alphabet with the common partition* $\Sigma_f = \Sigma_n \dot\cup \{\mathrm{f}\} = \Sigma_c \dot\cup \Sigma_{uc} = \Sigma_o \dot\cup \Sigma_{uo}$ *and a plant* $L_f \subseteq \Sigma_n^*\{\epsilon, \mathrm{f}\}\Sigma_n^*$ *with persistent fault that does not conflict with* Σ_n^*. *For a nominal controller* H_n *admissible w.r.t.* $L_f \cap \Sigma_n^*$, *assume that the local closed loop* $K_{\mathrm{loc}} := (\mathrm{pre}\, L_f) \cap H_n$ *is diagnosable and does not deadlock. If a post-fault-detection controller* H_d *satisfies conditions [A1]–[A3], then the overall controller* H_f *defined by equation (24) is admissible to both* L_f *and* $L_f \cap \Sigma_n^*$. *Provided that the degraded specification* E_d *satisfies (22), the closed loop satisfies the bounds*

$$L_n \cap H_n \subseteq L_f \cap H_f \subseteq (L_n \cap H_n) \cup E_d\,.$$

Proof. Regarding the switching condition Eq. (23), we make the following preliminary observations. Let $s,\ t \in \Sigma_f^*$ and $\sigma \in \Sigma_f$. Then:

$$s \in C \quad\Rightarrow\quad st \in C\,, \tag{25}$$

$$s \notin C,\ s\sigma \in C \quad\Rightarrow\quad s\sigma \in C \cap H_n \cap H_d\,, \tag{26}$$

$$\mathrm{p_o}\, s = \mathrm{p_o}\, t \quad\Rightarrow\quad (s \in C \leftrightarrow t \in C)\,, \tag{27}$$

$$\mathrm{p_o}\, s = \epsilon \quad\Rightarrow s \notin C\,. \tag{28}$$

With (25)–(28) in place, each of the conditions [H0]–[H2] can be established for H_f as a consequence of the respective property present for H_n and, via [A1], for H_d. Regarding the relationship between T and C, we observe with [A3] that for any $s \in (\mathrm{pre}\, L_f) \cap (\mathrm{pre}\, H_f)$:

$$s \in C \quad\Leftrightarrow\quad s \in T\Sigma_f^*\,, \tag{29}$$

$$s \notin C \quad\Rightarrow\quad s \in K_{\mathrm{loc}}\,. \tag{30}$$

Condition [H3] for admissibility of H_f to L_f can then be established as a consequence of the above two implications; for [H4] we also refer to diagnosability and [A3]. Admissibility of H_f to the nominal plant $L_n := L_f \cap \Sigma_n^*$ follows by $H_f \cap \Sigma_n^* = H_n$ and admissibility of H_n to L_n. Regarding the lower closed-loop bound, observe $L_n \cap H_n = L_f \cap \Sigma_n^* \cap H_n = L_f \cap \Sigma_n^* \cap H_f \subseteq L_f \cap H_f$, where the second equality refers to the discussion below Eq. (24). For the upper closed-loop bounds pick an arbitrary $s \in L_f \cap H_f$. [Case 1] If $s \in \Sigma_n^*$, we conclude $s \notin C$, and, hence, $s \in L_n \cap H_n$. [Case 2] If $s \notin \Sigma_n^*$, we distinguish two further cases. [Case 2a] If the fault was detected, i.e. $s \in T\Sigma_f^*$, we conclude $s \in C$ and, hence, $s \in H_d$. Therefore, referring to [A2], $s \in E_d$. [Case 2b] If, on the other hand, the fault was

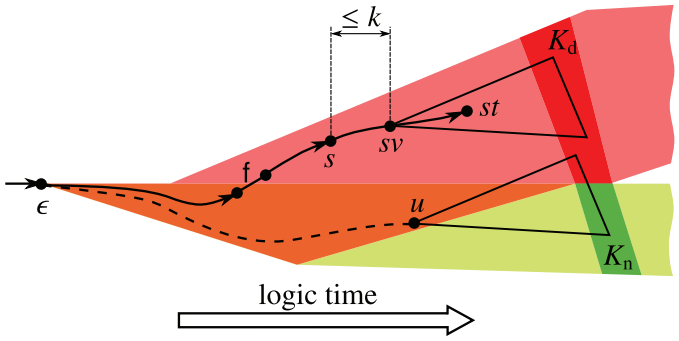

Fig. 7. Fault tolerant behaviour in the sense of Definition 11.

not detected, i.e. $s \notin C$, we have $s \in H_n$ and, thus, $s \in K_{loc}$. By diagnosability and liveness of K_{loc}, we extend s by t such that $st \in K_{loc} \cap T$. With (22) we conclude that $s \in E_d$, to establish the upper closed-loop bound. □

An alternative approach to design this over-all control scheme is to use the nominal closed-loop behaviour as the nominal specification; i.e., $E_n = L_n \cap H_n$. Then, H_f is obtained by solving the synthesis problem with input data L_f and $E_f = E_n \dot\cup E_d$, with subsequent extraction of H_d in compliance with the switching Eq. (24).

6. 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 either the nominal behaviour or the nominal specification. In the following we report on a framework developed in Wen et al. (2008a,2014); Wen et al. (2008b) which addresses recovery in terms of language convergence and variations thereof (Kumar et al., 1993; Willner & Heymann, 1995).

We begin our discussion with an adaption of *weak fault tolerance*, a closed-loop property originally defined by Wen et al. (2008b). For a presentation in the setting of the present paper and in contrast to the referenced literature, we use an explicit fault event and refer to Remark 1 for a formally non-blocking model.

Definition 11. A complete fault-accommodating behaviour $K_f \subseteq \Sigma_f^*$ with nominal part $K_n := K_f \cap \Sigma_n^*$ is *weakly fault tolerant*, if there exists a non-negative integer k, such that for all $s, t \in \Sigma_f^*$, $|t| \geq k$, with

$$s \in \operatorname{pre} K_f \cap \Sigma_n^* f \Sigma_f^*\,, \quad st \in \operatorname{pre} K_f\,, \tag{31}$$

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

$$K_f/sv \subseteq K_f/u\,. \tag{32}$$

The above definition requires that after a bounded delay any post-fault event sequence is consistent with some past from the nominal behaviour. For the illustration by Fig. 7, we use the union composition proposed in Section 4 with $K_f = K_n \dot\cup K_d$ and K_n the nominal component (solid green), K_d the degraded component (solid red), and with the corresponding prefixes in the respective light colour. The degraded sequence s, by Definition 11, with sufficiently long future t is seen to pass a configuration $sv < st$ from which on the possible future K_f/sv (triangular region right of sv) complies with the possible future K_f/u (triangular region right of u) of some nominal sequence u. As indicated by the sketch, the system may after recovery again be subject to the fault. Thus, this notion of fault tolerance also applies to *recurrent faults*, i.e., to faults that can occur arbitrarily often.

On the other hand, for persistent faults $K_f \subseteq \Sigma_n^*\{\epsilon, f\}\Sigma_n^*$, Eq. (32) is equivalent to

$$K_f/sv \subseteq K_n/u\,, \tag{33}$$

and, referring to Wen et al. (2008b), Theorem 3, weak fault tolerance implies

$$K_n/\Sigma_n^* \quad \Leftarrow \quad K_f/(\Sigma_n^* f)\,. \tag{34}$$

By Inclusion (32), weak fault tolerance imposes an upper bound on the post-fault behaviour and, in this sense, is interpreted as the recovery of safety properties. For the recovery of liveness properties in the reading of the present paper, Wen, Kumar, and Huang (2014) propose a stronger notion of fault tolerance by replacing Inclusion (32) by equality:

$$K_f/sv = K_f/u\,. \tag{35}$$

For an overall design, fault tolerance in the sense of Definition 11 and its variations is required as an additional closed-loop property. In Wen et al. (2008b), the authors present procedures to test whether a closed-loop candidate is fault tolerant and whether there exists a supervisor that achieves the candidate closed-loop behaviour. For the weak variant with Inclusion (32), the procedures are based on a characterisation in terms of language convergence. For the stronger variant, Eq. (35) is observed to be equivalent to $sv\,[\equiv_{K_f}]\,u$. In consequence and interpreted for a minimal realisation, the condition can be conveniently characterised by a state attraction property. In the reading of the present paper, the discussion addresses all of the admissibility conditions [H0]–[H6] except for the completeness properties [H3] and [H5]. The synthesis of fault-tolerant controllers is treated in Wen et al. (2014) under the additional assumption that all events including the fault are observable. Regarding optimality, a nearby objective is to maximise the pre-fault behaviour and to minimise the recovery time. However, the authors demonstrate by example that a maximal achievable pre-fault behaviour in general does not exist. This observation is closely related to the fact that language convergence due to the convergence bound is not retained under arbitrary union. For this reason, 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. In this setting, a procedure for the computation of an optimal solution is established.

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.

Definition 12. Given a fault-accommodating plant $L_f \subseteq \Sigma_f^*$, with nominal part $L_n = L_f \cap \Sigma_n^*$, an admissible controller H_f is *fault tolerant* if it guarantees the following properties for the closed loop $K_f = L_f \cap H_f$ with 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 $s \in K_f \cap \Sigma_n^* f \Sigma_n^*$ there exists a partition
$s = u_1 v_1 u_2 v_2 \dots u_k v_k f t$ such that
$u = u_1 u_2 \dots u_k \in \operatorname{pre} E_n$ and
$v = v_1 v_2 \dots v_k t \in E_d$,
[P3] $E_f \Leftarrow K_f/(\Sigma_n^* f)$.

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. Note that both, [P1] and [P2], can be represented as a language-inclusion specification while [P3] relates the approach to weak fault tolerance in the sense of

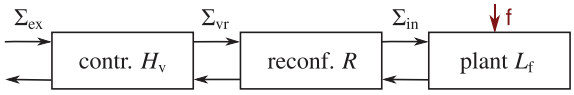

Fig. 8. Fault-hiding control architecture.

Definition 11. For the case that all events are observable, Sülek and Schmidt (2014) give a complete and sound algorithm for the synthesise of admissible controllers that are fault-tolerant in the sense of Definition 12. It is also demonstrated how the algorithm can be used for a subsequent repair procedure to retain the nominal specification and to address recurrent faults.

Remark 13. As a general comment on the concept of language convergence and the above variations, recall that they impose a bound on the number of events allowed until satisfactory behaviour is attained. This can be restrictive, e.g., when the plant is composed from multiple components which possess independent liveness properties. Here, a healthy component my generate an unbounded number of events to eventually complete a task which in turn is required to attain a recovery state. However, when dropping the bound, convergence of $*$-languages becomes in general a too weak requirement; e.g., without the bound, any $*$-language $L\subseteq\Sigma^*$ converges to $M=N^*$, $N\subseteq\Sigma^*$, since $\Sigma^*M=\Sigma^*$. Considering nonterminating processes, an alternative here is to model the plant by the corresponding ω-language $\mathcal{L}=\lim L$, i.e., the set of infinite strings with infinitely many prefixes in L. In this setting, the requirement of *eventual convergence* can be conveniently expressed as a language-inclusion specification

$$\lim L \subseteq \lim(\Sigma^* M)\,. \tag{36}$$

The corresponding synthesis problem is studied by Thistle and Wonham (1994) and Thistle and Lamouchi (2009).

7. Fault-hiding approach

With the design strategy of *fault hiding*, one begins with a given fault-accommodating model and a nominal controller. 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. In turn, the feedback provided by the actual plant is re-interpreted to generate feedback accepted by the nominal controller; see Fig. 8. The reconfiguration mechanism is meant to pretend nominal plant behaviour to the nominal controller while imposing fault-tolerant control on 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); Steffen (2005). For discrete-event systems, a fault-hiding approach is provided by Wittmann et al. (2013).

At the first stage, the nominal controller $H_n\subseteq\Sigma_n^*$ and the plant $L_f\subseteq\Sigma_f^*$ need to be decoupled. Since in our modelling framework shared events are synchronised, this requires a uniform renaming of events. However, renaming all events would limit the discussion on how the reconfiguration mechanism should affect the nominal controller by providing virtual feedback. We therefore distinguish *internal events* Σ_{in} and *external events* Σ_{ex}, i.e., $\Sigma_n=\Sigma_{in}\dot\cup\Sigma_{ex}$. Internal events are shared between plant and nominal controller, while external events are exclusively processed by the controller. Then, decoupling amounts to the renaming of internal events and is represented by a map $h:\Sigma_n^*\to\Sigma_{ex}^*\cup\Sigma_{vr}^*$ that encodes a bijective translation from internal events Σ_{in} to newly introduced distinct *virtual events* in Σ_{vr}, $\Sigma_{vr}\cap\Sigma_f=\emptyset$. Technically, the behaviour of the virtualised nominal controller composed with the plant is the shuffle product $L_f\|h(H_n)$. For the design of the reconfiguration mechanism $R\subseteq(\Sigma_{in}\cup\Sigma_{vr})^*$ the latter composition is given, and, hence is formally considered the plant, with $\Sigma_{in}\cap\Sigma_c$ and $\Sigma_{vr}\cap\Sigma_{uc}$ as controllable events and $\Sigma_{vr}\dot\cup\Sigma_{in}$ as observable events. A key feature of the proposed event renaming scheme is that the fault-accommodating specification $E_f\subseteq(\Sigma_f\cup\Sigma_{vr})^*$ still relates original plant events to external events and thereby ensures the intended semantics. Note that we may either assume $E_f=\mathrm{p}_f^{-1}\,\mathrm{p}_f E_f$ for the natural projection $\mathrm{p}_f:(\Sigma_f\cup\Sigma_{vr})^*\to\Sigma_f^*$, or encode specific requirements regarding the virtual events in E_f. Such requirements may express that R must pass on events one-by-one as their virtual counterpart unless the fault occurred. This is referred to as the *inactivity condition*.

The synthesis of R can be carried out according to the common admissibility criteria as presented in Section 3 applying the procedures from the nominal case. With an admissible reconfiguration mechanism R, an overall fault-accommodating controller $H_f\subseteq\Sigma_f^*$ can be obtained by projecting $h(H_n)\|R$ to Σ_n^* with subsequent inverse projection for a formal interpretation w.r.t. Σ_f^*. The cited literature includes a discussion on the admissibility of H_f and thereby interprets the result in the context of the naive approach to fault-tolerant control. Note that, if E_f encodes the inactivity condition, [H5] and [H6] are satisfied by construction and do not need to be addressed by a specific synthesis algorithm.

The particular challenge addressed by Wittmann et al. (2013) is the design of a reconfiguration mechanism that satisfies relevant admissibility criteria universally for *any* solution of the nominal control problem. This is of interest for applications in which the nominal controller is not known in terms of a formal model but only in terms of a verbal specification and/or hand-written PLC code. Technically, a solution to this synthesis problem needs to somehow eliminate the universal quantification over infinitely many formal plants parameterised by admissible nominal controllers. This is achieved by a tailored form of abstraction-based control. Referring to the minimally restrictive nominal closed-loop behaviour, universal quantification can be expressed by an upper bound in conjunction with the structural properties [K0]–[K3] possessed by any nominal closed-loop behaviour. We state the core result of this discussion for prefix-closed plant models in form of conditions [M1]–[M7]; see Wittmann (2014) for additional considerations that address not necessarily prefix-closed plant models.

Definition 14. Given the models $L_f\subseteq\Sigma_f^*$ and $L_n=L_f\cap\Sigma_n^*$ and the specifications $E_f\subseteq\Sigma_f^*$ and $E_n=E_f\cap\Sigma_n^*$, construct the supremal nominal controller $H^\uparrow\subseteq\Sigma_n^*$ and denote its virtualisation by $H_v^\uparrow:=h(H^\uparrow)\subseteq(\Sigma_{ex}\cup\Sigma_{vr})^*$. The following conditions are imposed on the a candidate closed loop $K\subseteq L_f\|H_v^\uparrow$:

- [M1] K is controllable w.r.t. $L_f\parallel H_v^\uparrow$ and the uncontrollable events $\Sigma_{ex}\cup h(\Sigma_c)\cup\Sigma_{uc}\cup\{\mathrm{f}\}$.
- [M2] K is prefix-normal w.r.t. $\mathrm{pre}\,(L_f\parallel H_v^\uparrow)$ and the observable events $\Sigma_{vr}\cup\Sigma_{in}$.
- [M3] K is relatively closed w.r.t. $L_f\parallel H_v^\uparrow$
- [M4] K is $(\Sigma_f-\Sigma_{ex})$-complete, i.e., $(\forall s\in\mathrm{pre}\,K\;\exists\sigma\notin\Sigma_{ex},\,t\in\Sigma_{ex}^*)[\,st\sigma\in\mathrm{pre}\,K\,]$.
- [M5] K is weakly sensor-event consistent, i.e., $(\forall s\in\mathrm{pre}\,K)[\,(\mathrm{p}_{vr}s)h(\Sigma_{uc})\cap(\mathrm{pre}\,h(L_n))\neq\emptyset \Rightarrow s(\Sigma-h(\Sigma_c))^*h(\Sigma_{uc})\cap\mathrm{pre}\,K\neq\emptyset\,]$
- [M6] K operates $H_v^\uparrow$ within $h(L_n)$, i.e., $\mathrm{p}_{vr}K\subseteq\mathrm{p}_{vr}\,\mathrm{pre}\,h(L_n)$.
- [M7] K satisfies the inclusion specification $K\subseteq E_f$.

If conditions [M1]–[M7] are satisfied, then the reconfiguration dynamics can be extracted from K by $R:=\mathrm{p}_o^{-1}\,\mathrm{p}_o((\mathrm{pre}\,K)\Sigma_{uc}^*)$ with event attributes as indicated above. Results reported by Wittmann et al. (2013) include admissibility of any formal overall

fault-accommodating controller H_f obtained from $R||h(H_n)$ with an arbitrary solution H_n to the nominal control problem. All properties [M1]–[M7] are retained under arbitrary union and an according synthesis procedure based on Moor et al. (2012) is elaborated in Wittmann (2014).

Summary

This paper provides a technical overview on fault tolerance for 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 concise and homogeneous notation, for a common interpretation in the context of the naive approach, where fault-tolerance is addressed by additional closed-loop requirements imposed on top of the common conditions of controllability and observability. Except for post-fault recovery, the properties discussed in this paper are retained under arbitrary union and the synthesis of a minimally restrictive controller can be implemented as a variation of the established procedures for supervisory control under partial observation. For post-fault recovery, stated in terms of language convergence or variations thereof, a supremal achievable closed-loop behaviour does not exist in general. Here, the referenced literature provides sound and complete procedures for the computation of a fault-tolerant controller under the assumption that all events including the fault are observable. Future research could address the general case of partial observation, where the synthesis problem for a related state attraction property has been recently solved by Schmidt and Breindl (2014). Additional insight could be gained by addressing fault-tolerance for languages of infinite strings, with a notion of eventual convergence that can be expressed as an upper-bound language-inclusion specification.

References

Baier, C., & Kwiatkowska, M. Z. (2000). On topological hierarchies of temporal properties. *Fundamenta Informaticae, 41*, 259–294.

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

Bourdon, E., Lawford, M., & Wonham, W. M. (2005). Robust nonblocking supervisory control of discrete-event systems. *IEEE Transactions on Automatic Control, 50*, 2015–2021.

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

Cassandras, C. G., & Lafortune, S. (2008). Introduction to Discrete Event Systems (2nd ed.). Springer.

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

Cury, J., & Krogh, B. (1999). Robustness of supervisors for discrete-event systems. *IEEE Transactions on Automatic Control, 44*, 376–379.

Dubrova, E. (2013). Fault-Tolerant Design. Springer.

Girault, A., & Rutten, R. (2009). Automating the addition of fault tolerance with discrete controller synthesis. *Formal Methods in System Design, 35*(2), 190–225.

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

Jiang, J., & Yu, X. (2012). Fault-tolerant control systems: A comparative study between active and passive approaches. *Annual Reviews in Control, 36*, 60–72.

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

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

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

Manna, Z., & Pnueli, A. (1990). A hierarchy of temporal properties. In *Proceedings of the 9th ACM symposium on principles of distributed computing* (pp. 377–408).

Moor, T. (2015). Fault-tolerant supervisory control. In *Proceedings of the 5th IFAC workshop on dependable control of discrete systems (DCDS)*. http://www.sciencedirect.com/science/article/pii/S2405896315007223 (accessed: 2016-01-05).

Moor, T., Baier, C., Yoo, T.-S., Lin, F., & Lafortune, S. (2012). On the computation of supremal sublanguages relevant to supervisory control. In *Proceedings of the 11th International workshop on Discrete Event Systems (WODES)* (pp. 175–180).

Moor, T., & Schmidt, K. W. (2015). Fault-tolerant control of discrete-event systems with lower-bound specifications. In *Proceedings of the 5th IFAC workshop on dependable control of discrete systems (DCDS)*.

Nke, Y., & Lunze, J. (2011a). A fault modeling approach for input/output automata. In *Proceedings of the 18th IFAC world congress* (pp. 8657–8662).

Nke, Y., & Lunze, J. (2011b). Online control reconfiguration for a faulty manufacturing process. In *Proceedings of the 3rd IFAC workshop on dependable control of discrete systems (DCDS)* (pp. 19–24).

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

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

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

Park, S. J., & Lim, J. T. (1998). Robust and fault-tolerant supervisory control of discrete event systems with partial observation and model uncertainty. *International Journal of Systems Science, 29*, 953–957.

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

Ramadge, P. J., & 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 *Proceedings of the 44th IEEE international conference on decision and control* (pp. 3493–3498).

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

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

Steffen, T. (2005). Control reconfiguration of dynamical systems: Linear approaches and structural tests. *LNCIS 320*. Springer-Verlag.

Sülek, A. N., & Schmidt, K. W. (2013). Computation of fault-tolerant supervisors for discrete event systems. In *Proceedings of the 4th IFAC workshop on dependable control of discrete systems (DCDS)* (pp. 115–120).

Sülek, A. N., & Schmidt, K. W. (2014). Computation of supervisors for fault-recovery and repair for discrete event systems. In *Proceedings of the 12th International workshop on discrete event systems (WODES)* (pp. 428–438).

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

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

Wen, Q., Kumar, R., & Huang, J. (2008a). Synthesis of optimal fault-tolerant supervisor for discrete event systems. In *Proceedings of the American control conference* (pp. 1172–1177).

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

Willner, Y. M., & Heymann, M. (1995). Language convergence in controlled discrete-event systems. *IEEE Transactions on Automatic Control, 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., & Moor, T. (2012). Fault-tolerant control of discrete event systems based on fault-accommodating models. In *Proceedings of the 8th IFAC symposium on fault detection, supervision and safety of technical processes (SAFEPROCESS)* (pp. 854–859).

Wittmann, T., Richter, J., & Moor, T. (2013). Fault-hiding control reconfiguration for a class of discrete-event systems. In *Proceedings of the 4th IFAC workshop on dependable control of discrete systems (DCDS)*.

Zaytoon, J., & Lafortune, S. (2013). Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control, 37*, 308–320.

Thomas Moor received his PhD degree (Dr.-Ing.) in 1999 from the University of the Federal Armed Forces Hamburg. From 2000 to 2003 he was a research fellow with the Research School of Information Sciences and Engineering at the Australian National University. Since 2003, he holds a professorship at the Lehrstuhl für Regelungstechnik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. His research interests include the control of discrete-event systems and hybrid systems, hierarchical and/or modular control systems, control system abstraction and fault-tolerant control. He serves on the Editorial Board of the Journal of Discrete Event Dynamic Systems and co-chaired the Workshop on Discrete Event Systems when it took place in Berlin 2010. He is maintainer and principle developer of the discrete-event systems software library libFAUDES, with a particular focus on supervisory control in an industrial application context.