Fault-Tolerant Control of Discrete-Event Systems with Lower-Bound Specifications

Thomas Moor * **Klaus Werner Schmidt** **

* *Lehrstuhl für Regelungstechnik*
Friedrich-Alexander Universität Erlangen-Nürnberg, Germany
(e-mail: lrt@fau.de)
** *Mechatronics Engineering Department*
Çankaya University, Ankara, Turkey
(e-mail: schmidt@cankaya.edu.tr)

Abstract: Fault-tolerant control addresses the control of dynamical systems such that they remain functional after the occurrence of a fault. To allow the controller to compensate for a fault, the system must exhibit certain redundancies. Alternatively, one may relax performance requirements for the closed-loop behaviour after the occurrence of a fault. To achieve fault tolerance for a hierarchical control architecture, a combination of both options appears to be advisable: on each individual level of the hierarchy, the controller may compensate the fault as far as possible, and then pass on responsibility to the next upper level. This approach, when further elaborated for discrete-event systems represented by formal languages, turns out to impose a hard lower-bound inclusion specification on the closed-loop behaviour. The present paper discusses the corresponding synthesis problem and presents a solution.

Keywords: Discrete-event systems, supervisory control, fault-tolerant control, hierarchical control.

1. INTRODUCTION

The aim of fault-tolerant control is the continuation of the system operation in case of a fault occurrence. Hereby, it is possible to make use of system redundancies or to allow a degradation of the system performance after encountering a fault. In this paper, the stated options are interpreted in the context of a hierarchical control architecture. We argue that, for each level of the hierarchy, a controller should use redundancies to compensate the effects of the fault whenever possible. When compensation becomes impossible, the fault should be made explicit for compensation by the next higher level, and so forth.

For discrete-event systems represented by formal languages, the described strategy can be formalised as a supervisory control problem with a hard lower-bound inclusion specification (such as the fault-free behaviour) and an upper-bound inclusion specification, that can be relaxed if otherwise no solution exists (in case of uncontrollable behaviour after a fault). This contrasts the well studied problem with a hard upper bound and the option to relax the lower bound. We note that the idea of relaxing the upper bound is solved by Lafortune and Chen (1990) for the prefix-closed case using the *infimal closed controllable superlanguage*. The authors also remark that for the general case of non-closed languages the infimum fails to be controllable. However, in the present paper, we are interested in non-closed languages for the specific situation of fault-tolerent control and propose a non-infimal but sensible solution for the problem at hand. Our approach retains regularity and we indicate how it can be implemented for finite automata representations.

Several approaches for fault-tolerance are proposed in the existing discrete-event systems literature, and we give selected references relevant to the present paper. Paoli and Lafortune (2005); Paoli et al. (2011) consider fault detection by a diagnoser and switching to a different supervisor for each fault before the desired system behaviour is violated. In (Wen et al., 2008) it is proposed that the system behaviour shall converge to the nominal system behaviour after a fault occurrence, whereas Kumar and Takai (2012) determine necessary and sufficient conditions for controller reconfiguration in the case of faults. Fault-accommodating models are used in Wittmann et al. (2013) to represent the fault and to develop a fault-hiding control architecture similar to the well established approach for continuous systems. The computation of supremal fault-tolerant sublanguages is suggested by Sülek and Schmidt (2013) for systems where certain events can not longer occur. Sülek and Schmidt (2014) propose a method for converging to a desired behaviour under fault. It is common to all of the above approaches that one needs to explicitly provide a formal representation of some desirable (but potentially degraded) behaviour that is to be achieved after a fault occurrence. In the present contribution, we propose a systematic way to derive this design parameter from the nominal specification and a fault-accommodating plant model. Motivated by hierarchical control architectures, our controller indicates any effects of the fault that it cannot compensate by a distinguished event.

The paper is organised as follows. In Section 2, we provide the relevant notation. Section 3 discusses the classical supervisory control method. Section 4 illustrates fault-accommodating models by example and points out limitations of this approach. The main idea and method of relaxing the upper-bound specification language for fault tolerance is developed in Section 5 and applied to an example in Section 6.

2. PRELIMINARIES AND NOTATION

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., $s = (\mathrm{pre}_n\, s)(\mathrm{suf}_n\, s)$.

A *formal language* (or short a *language*) over Σ is a subset $L \subseteq \Sigma^*$. The *prefix* of a language $L \subseteq \Sigma^*$ is defined by $\mathrm{pre}\, L := \{r \in \Sigma^* \mid \exists\, s \in L : r \leq s\}$. 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 prefix operator distributes over arbitrary unions of languages. However, for the intersection of two languages L and H, we have $\mathrm{pre}\,(L \cap H) \subseteq (\mathrm{pre}\, L) \cap (\mathrm{pre}\, H)$. If equality holds, L and H are said to be *non-conflicting*.

For two languages $K,\ M \subseteq \Sigma^*$, K is said to *converge* to M, denoted by $M \Leftarrow K$, if there is a non-negative integer n such that for each $s \in K$, there exists an $i \leq n$ such that $\mathrm{suf}_i\, s \in M$. For two languages $N,\ K \subseteq \Sigma^*$, let $K/N := \{t \mid \exists\, s \in N : st \in K\}$. Then, K is said to *converge to M after N* if $M \Leftarrow K/N$ (also referred to as *conditional convergence*).

For the *observable events* $\Sigma_\mathrm{o} \subseteq \Sigma$, the *natural projection* $\mathrm{p_o}\colon \Sigma^* \to \Sigma_\mathrm{o}^*$ 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 $\sigma \in \Sigma_\mathrm{o}$, or, if $\sigma \notin \Sigma_\mathrm{o}$, 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 $r \in \Sigma_\mathrm{o}^*$. When applied to languages, the projection distributes over unions, and the inverse projection distributes over unions and intersections. The prefix commutes with projection and inverse projection.

Given two languages $L,\ K \subseteq \Sigma^*$, and a set of *uncontrollable events* $\Sigma_\mathrm{uc} \subseteq \Sigma$, we say K is *controllable w.r.t. L*, if $(\mathrm{pre}\, K)\Sigma_\mathrm{uc} \cap (\mathrm{pre}\, L) \subseteq \mathrm{pre}\, K$. Controllability is retained under union.

An *automaton* is a tuple $G = (Q, \Sigma, \delta, q_\mathrm{o}, Q_\mathrm{m})$, with *state set* Q, *initial state* $q_\mathrm{o} \in Q$, *marked states* $Q_\mathrm{m} \subseteq Q$, and the partial *transition function* $\delta\colon Q \times \Sigma \to Q$ with its common extension to the domain $Q \times \Sigma^*$. We denote the *generated language* $\mathrm{L}(G) := \{s \in \Sigma^* \mid \delta(q_\mathrm{o}, s)!\}$ and the *marked language* $\mathrm{L_m}(G) := \{s \in \Sigma^* \mid \delta(q_\mathrm{o}, s) \in Q_\mathrm{m}\}$.

Given a language, define the equivalence relation $[\equiv_L]$ on Σ^* by $s'\,[\equiv_L]\,s''$ if and only if $(\forall\, t \in \Sigma^*)[\, s't \in L \leftrightarrow s''t \in L\,]$. Then there exists a state minimal automata representation of L with the equivalence classs of $[\equiv_L]$ as state set. If the latter is finite, L is called *regular*.

3. SUPERVISORY CONTROL

We give a concise review of the basic control problem introduced by Ramadge and Wonham (1987, 1989), in a variation that turns out convenient for the present paper.

Given an alphabet Σ, consider a language $L \subseteq \Sigma^*$ to represent the *plant* with *local behaviour* $\mathrm{pre}\, L$ (set of all event sequences that can be generated as time passes) and *accepted behaviour* L (to indicate task completion).

For the purpose of control, the alphabet is partitioned in *controllable events* and *uncontrollable events*, i.e., $\Sigma = \Sigma_\mathrm{c} \,\dot\cup\, \Sigma_\mathrm{uc}$. In the original literature, the controller is represented as a *causal feedback map* $f\colon \mathrm{pre}\, L \to \Gamma$ with the set of *control-patterns* $\Gamma := \{\gamma \mid \Sigma_\mathrm{uc} \subseteq \gamma \subseteq \Sigma\}$. In this paper, we represent f as a language H and impose three requirements to obtain a setting equivalent to *non-blocking supervisory control* from the cited literature [1].

Definition 1. Given $\Sigma = \Sigma_\mathrm{c} \,\dot\cup\, \Sigma_\mathrm{uc}$, a language $H \subseteq \Sigma^*$ is an *admissible controller* for the plant $L \subseteq \Sigma^*$, if

(H0) H is closed,

(H1) H is controllable w.r.t. L, and

(H2) L and H are non-conflicting. □

For the commonly studied control problem, we consider a plant $L \subseteq \Sigma^*$ and lower- and upper *language inclusion specifications* $A \subseteq \Sigma^*$ and $E \subseteq \Sigma^*$, in order to ask for an admissible controller H, such that the *accepted closed-loop behaviour* $K = L \cap H$ satisfies the closed-loop requirement $A \subseteq K \subseteq E$.

Definition 2. Given $\Sigma = \Sigma_\mathrm{c} \,\dot\cup\, \Sigma_\mathrm{uc}$, the *control problem* under consideration is parametrised by languages $\emptyset \neq A \subseteq E \subseteq L \subseteq \Sigma^*$. A controller $H \subseteq \Sigma^*$ *solves* the problem, if it is admissible to the plant L and if in addition

(H3) $A \subseteq L \cap H \subseteq E$. □

Except for differences in notation, a constructive solution has been presented by Ramadge and Wonham (1987, 1989). It is based on two technical results. First, it is observed that a language K can be obtained as the closed-loop behaviour $K = L \cap H$ with some admissible controller H if and only if K is itself controllable and relatively closed w.r.t. L. Second, controllability and relative closedness are retained under arbitrary union. Thus, given the upper bound E, there exists a unique supremal subset that is controllable and relatively closed:

$$K^\uparrow := \sup\{K \subseteq E \mid K \text{ is cntrl. and rel. closed w.r.t. } L\}. \quad (1)$$

In particular, $H := \mathrm{pre}\, K^\uparrow$ is admissible and we have $L \cap H = K^\uparrow \subseteq E$. If $K^\uparrow$ happens to also satisfy the lower bound specification $A \subseteq K^\uparrow$, then H solves the control problem. If, on the other hand, $K^\uparrow$ does not satisfy the inclusion $A \subseteq K^\uparrow$, then neither does any other achievable closed-loop behaviour and, thus, the control problem has no solution. If the parameters $A \subseteq E \subseteq L$ are regular, then so is $K^\uparrow$, and an automaton representation can be computed in polynomial time [2]. Thus, for practical problems, one can first compute a representation of $K^\uparrow$ and then test for $A \subseteq K^\uparrow$.

4. EXAMPLE

We provide a simple example to illustrate the well-known results presented so far and we utilise the example to outline how supervisory control is applied to hierarchical control architectures and how one may address fault tolerance in this context.

Consider a processing machine that interacts with its environment by receiving a workpiece, processing the workpiece, returning the workpiece, and so forth. At a suitable level of abstraction, the machine is represented by the automaton defined in Fig. 1, referring to the events g (get a workpiece), a (use tool A for processing), b (use tool B for processing), p (progress increment), d (processing complete), and x (workpiece exits

[1] In the original literature, the plant is represented by an automaton, and, thus, can itself be blocking. When using a single language to represent the plant, blocking can be modelled by a distinguished event.

[2] More precisely, the complexity is $O(n^2m^2)$, where n and m denote the state counts of automata representations of L and E, respectively.

machine). Regarding transport, a controller may choose how long to keep the workpiece in the machine until enabling x. Regarding processing, a controller can schedule the usage of the two tools by enabling a and b according to a particular recipe.

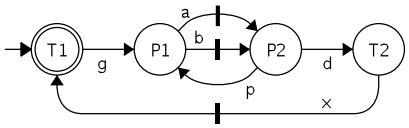

Fig. 1. plant behaviour L with contr. events {a, b, x}

The task is to design a controller that accepts external commands A and B to select a particular processing scheme and to provide external feedback X upon completion of a workpiece. The two language inclusion specifications defined in Fig. 2 relate the external events with the actual machine behaviour.Note that throughout this paper, we use the convention to interpret each individual automaton over the alphabet of those events that can actually occur. The overall specification E is then obtained by intersecting the inverse projections of each E_1 and E_2 to the full alphabet Σ = {g, a, b, p, d, x, A, B, X}.

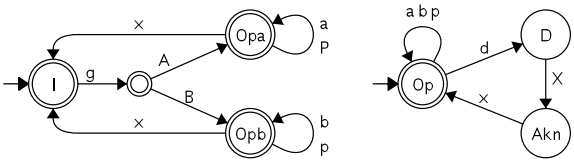

Fig. 2. upper bound specifications E_1 (left) and E_2 (right)

For the purpose of controller design, we regard the external interface events as controllable in order to allow the synthesis procedure to figure when to accept commands A or B and when to provide acknowledgment X according to the intended semantics; i.e., Σ_c = {a, b, x, A, B, X}. An automaton representation of the resulting supremal controllable and relatively closed sublanguage $K^{\uparrow}$ is shown in Fig. 3. As expected, both processing schemes are scheduled according to the external commands A and B, with acknowledgment X at completion. Thus, for practical purposes, we can accept the lower bound $A := K^{\uparrow}$ implied by the upper bound E. The closed-loop behaviour w.r.t. the external interface is given by the projection $L_o := p_o K^{\uparrow}$ with Σ_o :={A, B, X}; see Fig. (3).

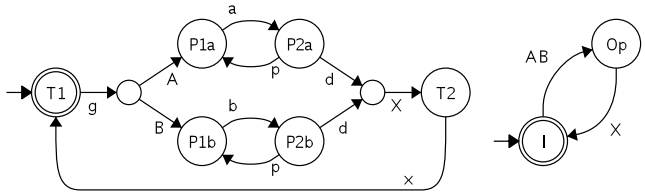

Fig. 3. closed loop $K^{\uparrow}$ (left) with external behaviour L_o (right)

The external interface can be put to use in the context of a larger manufacturing system made up from multiple processing machines. After equipping each individual machine with *low-level control* as in the above example, one may compose the respective external behaviours to obtain an abstraction of an overall plant. This abstraction can be used for the subsequent design of a controller that coordinates the individual machines. In principle, the *high-level controller* can be constructed by the same synthesis method as demonstrated in the above design of a low-level controller. However, special care must be taken regarding possible conflicts in the overall system, in particular when shared resources are represented by shared events. This issue has been discussed intensively with a range of practical results; e.g. (Schmidt et al., 2008; Wong and Wonham, 2000, 1996; Schmidt and Breindl, 2011; Komenda et al., 2012; Leduc et al., 2001). [3]

Turning the discussion towards fault tolerance, we assume that processing tool A is subject to a breakdown condition, referred to as *fault*. For the purpose of the present paper, we follow the approach of *fault-accommodating models* proposed by Wittmann et al. (2012). The authors suggest to represent the fault by a distinguished event $f \notin \Sigma$ and otherwise to proceed as usual [4]. For its simplicity, this approach seamlessly integrates with a wide range of hierarchical design methodologies.

Technically, the alphabet is extended $\Sigma_f := \Sigma \dot{\cup} \{f\}$ by the fault event {f}, and we use a language $L_d \subseteq \Sigma_f^*$ to represent the conditions under which the fault occurs and the effect it has for the future behaviour, i.e., we may assume $L_d \cap \Sigma^* = \emptyset$ and $\mathrm{pre}\, L_d \cap \Sigma^* \subseteq \mathrm{pre}\, L$. Then, the plant behaviour including the fault amounts to the union $L_f = L \cup L_d$, where the original model L is also referred to as the *nominal plant*. For our example, we use the fault-accommodating model L_f defined in Fig. 4.

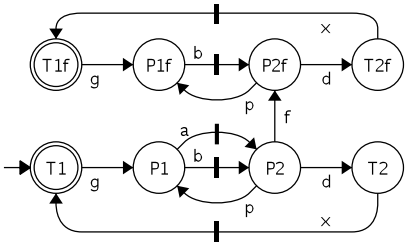

Fig. 4. fault-accommodating model L_f

It is readily verified that the nominal controller $H := \mathrm{pre}\, K^{\uparrow}$ conflicts with the fault-accommodating plant L_f. Therefore, we lift the original specification to the extended alphabet by $E_f := p_f^{-1} E$, with the set-valued inverse p_f^{-1} of the natural projection $p_f : \Sigma_f^* \to \Sigma^*$. Re-evaluation of the synthesis formulae with L_f and E_f as parameters yields the closed-loop behaviour $K_f^{\uparrow}$ with automata representation given in Fig. 5. By design, the controller $H_f := \mathrm{pre}\, K_f^{\uparrow}$ is admissible for L_f. In particular, L_f and H_f are non-conflicting – in this sense, H_f is a fault tolerant controller. However, we also observe the strict inclusions $K_f^{\uparrow} \subsetneq K^{\uparrow}$ and $L_{of} \subsetneq L_o$. In our design, the controller never uses tool A in oder to prevent a subsequent conflict caused by a possible breakdown. This is regarded a conservative strategy: for our application we would prefer the controller to utilise tool A until the fault actually occurs.

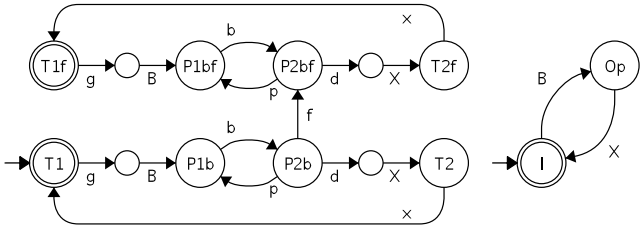

Fig. 5. fault tolerant $K_f^{\uparrow}$ (left) with ext. behaviour L_{of} (right)

[3] The processing machine example has been verified to be consistent for the purpose of abstraction based control with the method proposed by Moor (2014). Alternatively, one can extend the interface alphabet by g to satisfy the natural observer condition; see (Wong and Wonham, 2000; Schmidt et al., 2008).

[4] Wittmann et al. (2012) flag the fault event as unobservable and then apply the common approach for supervisory control under partial observation.

5. RELAXING THE UPPER BOUND

Given a control problem with lower bound A and upper bound E for the closed-loop requirement (H3) from the previous section, let $K^\uparrow$ denote the supremal closed-loop behaviour that satisfies the upper bound inclusion $K^\uparrow \subseteq E$ under admissible control; see (1). Recall that if $A \not\subseteq K^\uparrow$, the problem has no solution. In this case, $K^\uparrow$ can be interpreted as the

minimal relaxation of the lower bound A required to solve the problem while insisting on the upper bound E.

As long as $K^\uparrow$ still indicates some relevant behaviour (e.g. more than just the empty string), giving the upper bound E priority and setting $A := K^\uparrow$ is a reasonable approach for many application. However, as pointed out by the example in the previous section, there are also situations that suggest the converse strategy: if the original control problem has no solution, one may alternatively ask for the

minimal relaxation of the upper bound E to establish solvability while insisting on the lower bound A.

One may approach this question via the following infimum:

$$K^\downarrow := \inf\{K \supseteq A \mid K \text{ is cntrl. and rel. closed w.r.t. } L\}\,. \tag{2}$$

However, while relative closedness is retained under arbitrary intersection, controllability is not. Only in the special case when L is closed, the above infimum matches the *infimal closed and controllable superlanguage* which is itself controllable; see (Lafortune and Chen, 1990). For the purpose of this paper, prefix-closedness of L is considered a too restrictive prerequisite and we present an alternative approach.

Our discussion is based on a quantitative analysis for how the upper bound E restricts the closed-loop behaviour, which shall give us an indication of where to relax E such that the supremal closed-loop behaviour $K^\uparrow$ that satisfies E also satisfies the lower bound A. For an arbitrary sequence $s \in \operatorname{pre} L$ in the local plant behaviour, consider the following languages

$$L_s := L \cap (s\Sigma^*)\,, \tag{3}$$

$$E_s := E \cap (s\Sigma^*)\,, \tag{4}$$

$$K_s^\uparrow := \sup\{K \subseteq E_s \mid K \text{ is cntrl. and rel. closed w.r.t. } L_s\}\,. \tag{5}$$

The plant L_s and the upper bound specification E_s are obtained as variants of the original parameters L and E, respectively, under the additional assumption that event sequences generated by the plant will start with tracking s. By construction, the closure $H_s := \operatorname{pre} K_s^\uparrow$ is admissible for the plant L_s and will enforce the upper bound specification $E_s \subseteq E$. Moreover, provided that $K_s^\uparrow$ is nonempty, we must have $s \in \operatorname{pre} K_s^\uparrow$. If, for whatever reason, the plant indeed starts by tracking s, one may from then on exercise control H_s to enforce E. We summarise all sequences s with nonempty $K_s^\uparrow$ as the *target* set T,

$$T := \{s \mid K_s^\uparrow \neq \emptyset\}\,. \tag{6}$$

Once the plant has tracked a sequence s from the target T, enforceability of E is persistent under control H_s.

Proposition 3. For the control problem under consideration and two sequences $s, r \in \Sigma^*$ with $K_s^\uparrow$ as defined by (5), the following implication holds:

$$s \leq r \in \operatorname{pre} K_s^\uparrow \quad \Rightarrow \quad r \in T\,. \tag{7}$$

Proof. For $r \in \operatorname{pre} K_s^\uparrow$ with $s \leq r$, we consider the candidate $K := K_s^\uparrow \cap (r\Sigma^*)$. By the prerequisite on r, we have $r \in \operatorname{pre} K$. We now establish that K is an achievable closed-loop behaviour for the plant L_r that satisfies the upper bound $K \subseteq E_r$. For controllability of K, pick $t \in \operatorname{pre} K$ and $\sigma \in \Sigma_{\mathrm{uc}}$ such that $t\sigma \in \operatorname{pre} L_r$. For the case of $t < r$, we obtain $t\sigma \in \operatorname{pre} r \subseteq \operatorname{pre} K$. If not $t < r$, we must have $r \leq t$. Here, we note by $s \leq r \leq t\sigma$ that $t\sigma \in \operatorname{pre} L_s$ and $t \in \operatorname{pre} K_s^\uparrow$. Thus, controllability of $K_s^\uparrow$ implies $t\sigma \in \operatorname{pre} K_s^\uparrow$, and, we again obtain $t\sigma \in \operatorname{pre} K$. For relative closedness, observe that $(\operatorname{pre} K) \cap L_r \subseteq (\operatorname{pre} K_s^\uparrow) \cap L_s \cap (r\Sigma^*) = K_s^\uparrow \cap (r\Sigma^*) = K$. In summary, $K \subseteq E_r$ is indeed an closed-loop behaviour for the plant L_r. This implies $K \subseteq K_r^\uparrow$ and, thus, concludes the proof. □

The following proposition relates T to the upper bound $K^\uparrow$ of achievable closed-loop behaviours of the control problem under consideration.

Proposition 4. For the control problem under consideration and with $K^\uparrow$ and T defined by Eqs. (1) and (6), we have

$$\operatorname{pre} K^\uparrow \subseteq T\,, \tag{8}$$

where equality holds if and only if T is closed.

Proof. For the special case of $s = \epsilon$, we observe $K_s^\uparrow = K^\uparrow$. Then, implication (7) reads $[\, r \in \operatorname{pre} K^\uparrow \;\rightarrow\; r \in T\,]$, and, thus, the inclusion (8) is derived from Proposition 3. Regarding equality, closedness of T is obviously implied by $\operatorname{pre} K^\uparrow = T$. For the converse implication, assume that T is closed, and, thus satisfies (H0). Now pick an arbitrary $t \in \operatorname{pre} T$ and $\sigma \in \Sigma_{\mathrm{uc}}$ such that $t\sigma \in \operatorname{pre} L$. Then we can choose s such that $t \in \operatorname{pre} K_s^\uparrow$. As above, this implies $t\sigma \in \operatorname{pre} K_s^\uparrow$ for both cases $t < s$ and $s \leq t$. In particular, we conclude $t\sigma \in \operatorname{pre} T$ and have established controllability (H1) of T w.r.t. L. Likewise, non-conflictingness (H2) can be observed. Pick an arbitrary $t \in (\operatorname{pre} L) \cap T$ to then choose s such that $t \in \operatorname{pre} K_s^\uparrow$. Again, for both cases $t < s$ and $s \leq t$, we obtain $t \in \operatorname{pre}(L \cap T)$. Clearly, $T \subseteq E$, and E can be used as a controller that enforces the upper bound E. Supremality of $K^\uparrow$ then implies $T \subseteq K^\uparrow$. Taking prefixes on both sides together with the inclusion (8) implies equality $T = \operatorname{pre} K^\uparrow$. □

Thus, if T is closed, it solves the control problem with accepted closed-loop behvaiour $K^\uparrow$ and provides no further insight. More interestingly, if T fails to be closed, we have $\operatorname{pre} K^\uparrow \subsetneq \operatorname{pre} T$. In this situation, we propose to replace the implicit requirement $\operatorname{pre} A \subseteq \operatorname{pre} K^\uparrow$ imposed by the original problem parameters with the weaker requirement

$$\operatorname{pre} A \subseteq \operatorname{pre} T\,, \tag{9}$$

i.e., the lower bound A only needs to be consistent with the persistent possibility of enforcement of the upper bound E.

Now consider the candidate controller H defined by

$$M := \cup_{s \in T} \operatorname{pre} K_s^\uparrow\,, \tag{10}$$

$$N := \{s\sigma t \mid s \in M,\ s\sigma \notin M,\ \sigma \in \Sigma_{\mathrm{uc}} \text{ and } t \in \Sigma^*\}\,, \tag{11}$$

$$H := M \cup N\,. \tag{12}$$

The rationale for component M is to enforce the upper bound E, once the closed-loop evolves on a string that allows so. Recall that $s \in \operatorname{pre} K_s^\uparrow$ for $s \in T$ to observe $M = \operatorname{pre} T$. The rationale for component N is to allow the closed-loop to exit M, whenever controllability requires so. We give a formal statement of relevant properties of our candidate controller H.

Proposition 5. For the control problem under consideration we impose the additional requirement (9). Then the candidate controller H defined by (12), also referring to (5) and (6), is admissible for the plant L. Moreover, the accepted closed-loop behaviour $K = L \cap H$ satisfies the bounds

$$A \subseteq K \subseteq E \cup N. \tag{13}$$
$$A \cap (T\Sigma^*) \subseteq K \cap (T\Sigma^*) \subseteq E. \tag{14}$$

Proof. We make two preliminary observations from the definition of M and N. As a union of closed languages M is itself closed. Moreover, for all $r \in \operatorname{pre} N$ with $r \notin M$, we have $r\Sigma^* \subseteq N$. We now verify that H is an admissible controller for the plant L. To establish that H is closed (H0), pick any strings $r \in H$. If $r \in M$, we have $\operatorname{pre} r \subseteq M \subseteq H$ since M is closed. If $r \in N$ we can rewrite $r = s\sigma t$ with $s \in M$ and $s\sigma \notin M$. This implies $s\sigma \in N$ and $s\sigma \operatorname{pre} t \subseteq N \subseteq H$. Regarding prefixes of r shorter than $s\sigma$, we have $\operatorname{pre} s \subseteq M \subseteq H$. We now turn to controllability (H1). Let $r \in \operatorname{pre} H$ and $\sigma \in \Sigma_{\mathrm{uc}}$ such that $r\sigma \in \operatorname{pre} L$. For a proof by contradiction, assume that $r\sigma \notin H$. Then $r\sigma \notin M$ and $r\sigma \notin N$. By the definition of N, the conjunction implies $r \notin M$, and, by $r \in H$, we must have $r \in N$. Referring to the preliminary observation, we obtain $r\sigma \subseteq r\Sigma^* \subseteq N \subseteq \operatorname{pre} H$. This contradicts with the initial assumption, and we conclude $r\sigma \in \operatorname{pre} H$. To establish a non-conflicting closed loop (H2), pick any $r \in (\operatorname{pre} L) \cap H$. For the first case, assume that $r \in M$. Thus, by the definition of M, we have $r \in \operatorname{pre} K_s^{\uparrow}$ for some $s \in T$. In particular, there exists t such that $rt \in K_s^{\uparrow} \subseteq L_s \subseteq L$ and $rt \in K_s^{\uparrow} \subseteq H$. Thus, $r \in \operatorname{pre}(L \cap H)$. For the second case, assume that $r \notin M$, and, hence, $r \in N$. By the preliminary observation, this implies $r\Sigma^* \in N$. By $r \in \operatorname{pre} L$, we pick t such that $rt \in L$ and again obtain $r \in \operatorname{pre}(L \cap H)$. This concludes the proof of (H2). So far, we have established that H is indeed an admissible controller for L. Regarding the lower bounds, pick an arbitrary $s \in A$. With the prerequisite (9) we obtain $s \in \operatorname{pre} T$, and we choose t such that $st \in T$. Hence, $st \in \operatorname{pre} K_{st}^{\uparrow}$ and, in particular, $s \in \operatorname{pre} K_{st}^{\uparrow} \subseteq M \subseteq H$. With $s \in A \subseteq L$ we obtain $s \in K$ We turn to the upper bounds and pick an arbitrary $s \in K$. Consider the case where $s \in M$ and choose r such that $s \in \operatorname{pre} K_r^{\uparrow}$. In particular, we have $s \in r\Sigma^*$, and thus $s \in L_r$. Relative closedness of $K_r^{\uparrow}$ w.r.t. L_r then implies $s \in K_r^{\uparrow} \subseteq E_r \subseteq E$. As intended, the closed loop complies to E as long as it is controlled by M, and for this case both upper bounds are satisfied. For the second case, we have $s \notin M$, and, hence, $s \in N$. For the second upper bound we can in addition use $s \in T\Sigma^*$ and pick $v \leq s$ such that $v \in T$, and, hence, $v \in \operatorname{pre} K_v^{\uparrow}$. From the definition of N, we denote $r < s$ the longest prefix with $r \in M$ to write $s = r\sigma t \in N$. Since $T \subseteq M$, we must have $v \in M$, i.e., $v \leq r$. We apply Proposition 3 to obtain $r \in \operatorname{pre} K_v^{\uparrow}$ and, in turn, controllability of $K_v^{\uparrow}$ implies $r\sigma \in \operatorname{pre} K_v^{\uparrow} \subseteq M$. Thus, we have derived a contradiction and conclude, that once a closed-loop trajectory passes T, it is controlled by component M, which enforces E. □

For practical controller synthesis, one may first compute an automaton representation of T to then derive representations of N and the relaxed specification $E' = E \cup N$. The supremal achievable closed-loop behaviour can then be computed in the usual manner. Regarding a representation of T we note the following proposition.

Proposition 6. For the control problem under consideration and with T defined by Equation (6), we have for all s, $s' \in \Sigma^*$ with $s\,[\equiv_L]\,s'$ and $s\,[\equiv_E]\,s'$that $s \in T$ if and only if $s' \in T$.

Proof. Let s and s' be equivalent as required and assume that $s \in T$, i.e., $K_s^{\uparrow} \neq \emptyset$. In particular $s \in \operatorname{pre} L$ and, by equivalence, $s' \in \operatorname{pre} L$. Following (Moor, 2014), Lemma 8, we obtain

$$H' = \{r \mid r \in \Sigma^*,\ s' \notin \operatorname{pre} r\} \cup \{s't \mid st \in \operatorname{pre} K_s^{\uparrow}\}$$

as an admissible controller with $s' \in \operatorname{pre}(L_{s'} \cap H')$. Thus, we are left to show that it enforces the upper bound. Pick any t such that $s't \in L_{s'} \cap H'$. In particular $st \in \operatorname{pre} K_s^{\uparrow}$. With $s't \in L$ equivalence implies $st \in L$, and, we obtain by relative closedness $st \in K_s^{\uparrow} \subseteq E$. Observe again by equivalence $s't \in E$. This concludes the proof with $L_{s'} \cap H' \subseteq E_{s'}$. □

By the above proposition, a representation of T can be constructed by inspection of the state set of the product composition of automata representations of L and E. The overall complexity is of order $O(n^2m^2)$, where n and m denote the state counts of the representations of L and E, respectively.

6. EXAMPLE (CNT.)

We continue the example from Section 4. Recall that the closed-loop behaviour $K_{\mathrm{f}}^{\uparrow}$ achievable for the fault-accommodation plant L_{f} was a strict subset of the supremal closed loop $K^{\uparrow}$ for the nominal plant L. For the particular example, this is considered undesirable and we propose to relax the upper bound specification E according to the previous section.

We begin with computing the target T w.r.t. the original upper bound E and the fault-accommodating plant model L_{f}. For illustration purposes, Fig. 6 shows T (marked language) as a subset of $(\operatorname{pre} E) \cap (\operatorname{pre} L_{\mathrm{f}})$ (generated language). Note that, only when the fault occurs during processing mode A (state P), with a subsequent progress event p, the generated string escapes from $\operatorname{pre} T$.

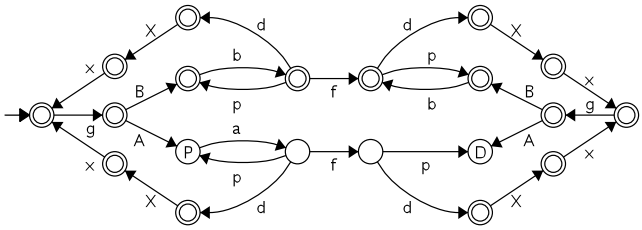

Fig. 6. target T for plant L_f and original upper bound E

Observe that $\operatorname{pre} K^{\uparrow} \subseteq \operatorname{pre} T$, and we can apply the method from the previous section to synthesise a controller for the lower bound specification $A = K^{\uparrow}$ by relaxing the upper bound E by $E' := E \cup N$ from Equation (13) in Proposition 5; see Fig. 7.

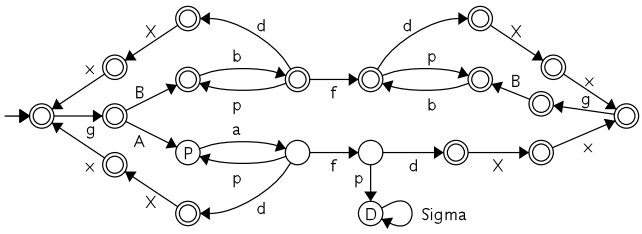

Fig. 7. relaxed upper bound $E' := E \cup N$

Computing the supremal achievable closed-loop behaviour K_{f}' and taking the projection to the high-level alphabet, we obtain the external behaviour L_{of}' given in Fig. 8.

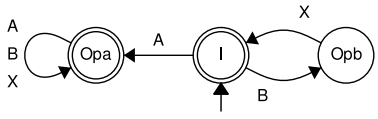

Fig. 8. external behaviour L_{of}' obtained by upper bound E'

Note that the external behaviour allows for either processing mode A or B. However, when selecting processing mode A, the component N from Equation (11) enables all events. Based on this model, no practical coordinating controller will ever select

mode B. We therefore introduce the distinguished event F to indicate externally when the fault has occurred and the original upper bound cannot be met anymore. This is achieved by using the relaxed upper bound $E'' := E \cup N''$ with

$$N'' := \{s\sigma \mathsf{F} t \mid s \in M,\ s\sigma \notin M,\ \sigma \in \Sigma_{\mathrm{uc}} \text{ and } t \in \Sigma^*\}, \qquad (15)$$

in place of the original component N. Re-evaluating the synthesis formulae, denote the closed-loop behaviour K_{f}'' and external behaviour L_{of}''; see also below Fig. 9.

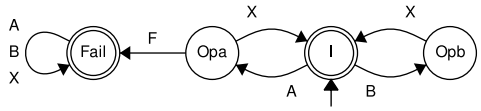

Fig. 9. ext. behaviour L_{of}'' obtained by upper bound $E'' = E \cup N''$

With this setting, both processing modes can be selected, and only when the fault takes effect, the external behaviour degrades. This is indicated by the high-level event F. However, no more desirable behaviour is exhibited by L_{of}'' after the fault.

As an option to shape the closed-loop behaviour K_{f}'' obtained so far, we propose a language convergence specification to recover the fault-tolerant behaviour $K_{\mathrm{f}}^{\uparrow}$ after the occurrence of F. Technically, we ask for an admissible controller H such that

$$K_{\mathrm{f}}^{\uparrow} \Leftarrow (K_{\mathrm{f}}'' \cap H)/(\Sigma^* \mathsf{F}). \qquad (16)$$

The synthesis problem for language convergence specifications is studied in (Willner and Heymann, 1995), including algorithmic decidability and a computational procedure for regular languages. An adaption for conditional convergence, as required for the present paper, has been presented in Sülek and Schmidt (2014) [5] . Note that the resulting controller H is guaranteed not to restrict the local closed-loop behaviour before the system passes the condition $\Sigma^*\mathsf{F}$. Technically, our lower bound $K_{\mathrm{f}}^{\uparrow}$ does not contain any F event. Hence, the additional controller H does not violate the lower bound $K_{\mathrm{f}}^{\uparrow}$.

The external behaviour of the final result for our example is given in Fig. 10. In a multiple machines setting, the coordinating controller can utilise both processing modes until the fault takes effect. From then on, mode A is disabled but mode B is still provided via the external interface.

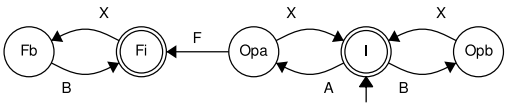

Fig. 10. ext. behaviour L_{of}''' with convergence specification

CONCLUSION

The objective of fault-tolerant control is to faithfully operate a plant which is subject to a fault. If a fault cannot be compensated by utilising plant redundancies, a fault-tolerant controller will allow for degraded performance. When performance specifications are given by language inclusions, one may insist on a given upper bound and accept the implied lower bound or one may insist in a given lower bound and accept the implied upper bound. While the first strategy is a common choice for discrete-event systems, we discuss the second strategy and insist in the nominal closed-loop behaviour as a lower bound when synthesising the fault-tolerant controller. We illustrate potential benefits in the context of hierarchical control by an example. There, the proposed approach amounts to a compensation of the fault whenever possible and a propagation of the remaining effects of the fault upwards in the control hierarchy.

[5] The proposed algorithm is of complexity $O(n^2 2^{2|m|})$, where n and m denote the state counts of automata representations of L and M, respectively.

REFERENCES

Komenda, J., Masopust, T., and Schuppen, J.H. (2012). Supervisory control synthesis of discrete-event systems using a coordination scheme. *Automatica*, 48(2), 247–254.

Kumar, R. and Takai, S. (2012). A framework for control-reconfiguration following fault-detection in discrete event systems. In *8th Symposium on Fault Detection, Supervision and Safety of Technical Processes*, 848–853.

Lafortune, S. and Chen, E. (1990). The infimal closed controllable superlanguage and its application in supervisory control. *IEEE Trans. Autom. Control*, 35, 398–405.

Leduc, R., Brandin, B., and Wonham, W. (2001). Hierarchical interface-based supervisory control: Serial case. *IEEE Conference on Decision and Control*, 4116–4121.

Moor, T. (2014). Natural projections for the synthesis of non-conflicting supervisory controllers. *Workshop on Discrete Event Systems (WODES)*.

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. (2011). Active fault tolerant control of discrete event systems using online diagnostics. *Automatica*, 47(4), 639–649.

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.

Schmidt, K. and Breindl, C. (2011). Maximally permissive hierarchical control of decentralized discrete event systems. *IEEE Trans. Autom. Control*, 56(4), 723–737.

Schmidt, K., Moor, T., and Perk, S. (2008). Nonblocking hierarchical control of decentralized discrete event systems. *IEEE Trans. Autom. Control*, 53(10), 2252–2265.

Sülek, A.N. and Schmidt, K.W. (2013). Computation of fault-tolerant supervisors for discrete event systems. In *4th 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.

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

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

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

Wittmann, T., Richter, J., and Moor, T. (2013). Fault-hiding control reconfiguration for a class of discrete event systems. *4th Workshop on Dependable Control of Discrete Systems*.

Wong, K.C. and Wonham, W.M. (1996). Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems*, 6, 241–306.

Wong, K. and Wonham, W. (2000). On the computation of observers in discrete-event systems. *Information Sciences*, 44, 173–198.