Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Efficient failure-recovering supervisors

N. Paape * **J.M. van de Mortel-Fronczak** * **L. Swartjes** **
M.A. Reniers *

* *Eindhoven University of Technology, Eindhoven, The Netherlands
(e-mail: n.paape@tue.nl, j.m.v.d.mortel@tue.nl, m.a.reniers@tue.nl)*

** *Vanderlande Industries B.V., Veghel, The Netherlands
(e-mail: lennart.swartjes@vanderlande.com)*

Abstract: Automated systems require controllers which guarantee machine safety and specified functionality even in case of occurring defects. In literature, several methods can be found for formally deriving a supervisor providing such guarantees, including the existence of failure recovery. In this paper, an extension is proposed so that the derived supervisor not only guarantees the existence of failure recovery, but also enforces a shortest path for it. To this end, a two-step procedure is defined for supervisor derivation, in which two algorithms are involved.

Keywords: Discrete-event systems, supervisory control, supervisor synthesis, fault tolerance, failure recovery

1 INTRODUCTION

Cyber-physical systems, like baggage handling, parcel sorting or warehousing solutions, require controllers which guarantee machine safety and specified functionality (liveness), even when they suffer from defects. As the complexity of these systems is increasing rapidly, this leads to more complex machine safety and functional specifications and a higher number of possible defects. As a result, it is becoming increasingly more difficult to design these controllers. To guarantee that machine safety is satisfied in these complex systems, controllers are often made overly restrictive, at the cost of the enabled system functionality.

The *supervisory control* layer guarantees machine safety and functionality in a *plant* (uncontrolled system). As mentioned in [Wonham and Cai, 2019] and [Zaytoon and Riera, 2017], a system and its supervisory controller — or in short supervisor — can generally be approximated as *discrete-event systems* (DES). The plant generates uncontrollable events at discrete instances in time (which the supervisor cannot prevent from occurring, typically sensor events), and the supervisor guarantees the required machine safety and functional specifications by enabling or disabling controllable events (which the supervisor can prevent from occurring, typically actuator events). Examples of machine safety and functional specifications in a cyber-physical parcel sorting system are, respectively: no packages are released to a conveyor which is full, and each package is eventually released to the conveyor.

As defects have a detrimental influence on the machine safety and functionality of the controlled system [Fritz and Zhang, 2018], maintaining both during defects is one of the major challenges in the design of a supervisor. The two types of defects which are dealt with in this paper are faults and failures, which are classified in [Gertler, 2014] and [Cho and Lim, 1998]. A *fault* is a malfunction in the plant which degrades system performance. For faults, any dangerous behavior can generally be prevented through the use of a *fault-tolerant supervisor* with fault-specific machine safety and functional specifications. However, for a subclass of faults — *failures* — the malfunction can result in a total system breakdown, meaning the system can end up in a *critical* (dangerous) state. Unlike with faults, reaching these dangerous states cannot be prevented, as important specified functionality during nominal system behavior would need to be disabled (e.g. a system will obviously never be in a dangerous state if it is never allowed to turn on at all). If prevention is not a possible solution, then instead, a supervisor should be designed which guarantees that after a failure the system returns to a *noncritical* system state. Such a supervisor is said to be a *failure-recovering supervisor*.

Many methods exist to derive fault-tolerant supervisors without taking failure recovery into account. For example, in [Reijnen et al., 2018], a design method is proposed based on an existing synthesis procedure which provides a fault-tolerant supervisor, using guards in extended finite-state automata (EFA) and by describing nominal and post-fault system specifications through state-based expressions. This method offers the following advantages: it uses structured modeling steps, the resulting state-based requirements intuitively follow from textual specifications, it can handle multiple faults, and it supports the modular addition of post-fault requirements.

Additionally, several methods for the design of failure-recovering supervisors have been proposed. [Wen et al., 2014] addresses failure recovery with *bounded convergence* to nominal system behavior (guaranteed recovery in a finite number of events). [Acar and Schmidt, 2015] introduces a similar method, but with bounded convergence to a specified desired failed system behaviour, after which the

system can be repaired back to the nominal specifications. In [Schuh and Lunze, 2016], a method is proposed for recovery to a desired final state using a diagnostic unit, and a reconfigurable controller. In [Andersson et al., 2011], a control method for manufacturing systems is presented in which restart states are used to return to nominal behaviour after a failure.

The purpose of this paper is to integrate the failure-recovering functionality in the method proposed in [Reijnen et al., 2018]. To this end, on top of the nominal and post-fault specifications, a new type of required behavior specification is proposed: recovery objectives. Based on these objectives, additional properties are defined which a failure-recovering supervisor should satisfy. The synthesis algorithm described in [Ouedraogo et al., 2011] is modified in such a way that it provides the maximally permissive failure-recovering supervisor. Subsequently, the failure-recovering functionality in this supervisor is streamlined, keeping only the shortest paths which accomplish the specified objectives.

The methods introduced in [Wen et al., 2014] and [Acar and Schmidt, 2015] most closely match our approach for failure recovery. However, there are a few key differences, which are important for application in cyber-physical systems:

Firstly, in cyber-physical systems it is often not possible to recover to the nominal system as done in [Wen et al., 2014] (e.g., if a component breaks down permanently). It is often required to recover to a desired failed system behaviour instead, as done in this paper and in [Acar and Schmidt, 2015].

Secondly, it is crucial that recovery is controllable (i.e., from every critical state a finite string of controllable events exists to a noncritical state). This is done in this paper and in [Wen et al., 2014], but not in [Acar and Schmidt, 2015]. Uncontrollable events are not guaranteed to ever occur, so if a recovery is uncontrollable, then the system could stay critical indefinitely.

Finally, the method introduced in this paper is the only one of the three in which convergence is unbounded. This is because cyber-physical systems often have components which can generate infinite strings of uncontrollable events, making it impossible to synthesize a supervisor with bounded convergence such as in [Wen et al., 2014] and [Acar and Schmidt, 2015]. In the method presented in this paper, infinite strings are allowed during critical system behaviour, as long as recovery is still controllable. Of course, this does mean that the safety of the system depends on the supervisor being able to actually enforce these controllable events.

The paper is structured as follows. First, preliminaries regarding modelling with DES are given in Section 2. In Section 3, the problem definition is given and the efficient failure-recovering supervisor is defined. In Section 4, algorithms are proposed for the derivation of such a supervisor and for achieving the desired efficiency. Finally, concluding remarks are presented in Section 5.

2 PRELIMINARIES

This section summarizes basic notations regarding modeling DES by extended finite automata (EFA).

2.1 Extended finite automata

An EFA is a 7-tuple $A = (L, D, \Sigma, E, L_0, D_0, L_m)$, where L is a finite set of locations, $L_0 \subseteq L$ is the set of initial locations, and $L_m \subseteq L$ is the set of marked locations.

When modeling a system, a finite number of variables with finite domains is used. The set of possible combinations of variable values, with p variables in the EFA, is described by data set $D = D^1 \times \cdots \times D^p$, with D^i representing the domain of the i-th variable, $1 \leq i \leq p$. The values of all variables at any given moment are represented by vector $d \in D$ of length p, $d = [d(1), \ldots, d(p)]$. The initial set of data values is $D_0 \subseteq D$.

Σ is a finite set of events. The set of events is partitioned into a set of controllable events Σ_c and a set of uncontrollable events Σ_u, which a supervisor respectively can and cannot disable. The set of fault events Σ_f is a subset of the set of uncontrollable events, and the set of failure events $\Sigma_{failure}$ is a subset of the set of fault events.

$E \subseteq L \times \Sigma \times \mathcal{G} \times \mathcal{U} \times L$ is a finite set of transitions, with $\mathcal{G}$ the set of guard predicates and $\mathcal{U}$ the set of data update functions. When discussing predicates, the notations of T for $true$ and F for $false$ are used. Each transition $e \in E$ with $e = (o_e, \sigma_e, g_e, u_e, t_e)$ is a 5-tuple, as defined below.

- $o_e \in L$ is the origin location of e.
- $\sigma_e \in \Sigma$ is the transition label of e.
- $g_e : D \to \{F, T\}$ is the enabling guard of e.
- $u_e : D \to D$ is the data update function of e.
- $t_e \in L$ is the terminal location of e.

Example EFAs shown in Sections 3 and 4 are represented graphically. For each EFA we assume the presence of a so-called location variable that always has the name of the current location as its value. The name of this variable is simply the name of the EFA. Updates of this location variable are implicit and not shown in the graphical representations. For an EFA with name A and a location with name L, the notation $A.L$ is the predicate that holds only if the automaton is in that location.

2.2 Plant, requirement and fault modeling

For synthesis of a supervisor, a model of the uncontrolled system and its requirement specification are needed. The uncontrolled system is modelled as an EFA $G = (L, D, \Sigma, E, L_0, D_0, L_m)$, and the requirement specifications are represented by $L_x \subseteq L$, which is the set of locations in G in which the requirements are not satisfied. Constructing such a monolithic model can quickly become problematic for more complex systems. Fortunately, there are methods to simplify this process. In this paper, the method proposed in [Markovski et al., 2010] is used, in which EFA G and forbidden states L_x are constructed out of smaller automata describing the systems components, and state-based expressions describing the requirement specifications. These state-based expressions are formulated as "*e needs Y*", which translates to events $e \in E$ is

only allowed when predicate Y evaluates to true. This syntax naturally extends to a set of events $\{e_1, \ldots, e_n\} \in E$ with the expression "$\{e_1, \ldots, e_n\}$ *needs* Y".

The set of all faults in a system is denoted by $\mathcal{F}$. The set $\mathcal{F}_{failure}$ of all failures is a subset of $\mathcal{F}$. To diagnose if a fault has occurred in the system requires detection, isolation, and identification of the fault. An overview of fault diagnosis approaches can be found in [Zaytoon and Lafortune, 2013]. However, which approach should be used is not in the scope of this paper. In this paper, it is assumed that all faults in the system can be diagnosed, with the occurrence of a fault being indicated by an observable fault event. Instead of using states to indicate the current status of the failure, Boolean variables are used. The status of a fault $f \in \mathcal{F}$ is indicated by Boolean fault variable v_f, which is true as long as fault f is in the system. This way, failure status is indicated consistently, regardless of how fault diagnosers are modelled. This is especially helpful when more complicated diagnoser models are used.

2.3 FA equivalence of EFA

As shown in [Sköldstam et al., 2007], each EFA can be transformed to an equivalent FA. Thus, supervisor properties which are defined for FAs, such as nonblocking and controllability, can also be defined for EFAs. An FA uses states, while an EFA uses locations and variables. However, for the FA equivalent of an EFA, each state $q = (l_q, d_q)$ is represented as a combination of a location $l_q \in L$ and the values of all variables at a given moment $d_q \in D$, with the set of all states defined as $Q = L \times D$. The set of initial states is defined as $Q_0 = L_0 \times D_0$. The set of marked states is defined as $Q_m = L_m \times D$. It is not necessarily true that every state in sets Q or Q_m can be reached from an initial state. Similarly, an EFA transition $e = (o_e, \sigma_e, g_e, u_e, t_e) \in E$ induces transitions in an equivalent FA as follows. For each vector $d \in D$, if $g_e(d) = T$ there is a transition from state (o_e, d) to state $(t_e, u_e(d))$ labelled with event σ_e.

To support reasoning about failure-recovering supervisors, three notions are defined: a path, reachability and a supervisor.

Definition 1. (Path). Let $G = (L, D, \Sigma, E, L_0, D_0, L_m)$ be an EFA. For $o, t \in L$, $d, d' \in D$ and $\sigma \in \Sigma$, by $(o, d) \xrightarrow{\sigma} (t, d')$ we denote that a transition $(o, \sigma, g, u, t) \in E$ exists such that $g(d) = T$ and $d' = u(d)$. A path from state q_1 to state q_{n+1} is an alternating sequence $(q_1, \sigma_1, q_2, \cdots, q_n, \sigma_n, q_{n+1})$ of states q_i and events σ_i such that $q_i \xrightarrow{\sigma_i} q_{i+1}$ for all $1 \leq i \leq n$. The length of path p, notation $|p|$, is the number of states in the path. $\mathcal{T}(p)$ denotes the number of transitions in path p. Notation $q \twoheadrightarrow q'$ denotes the existence of a path from q to q'. Notation $q \twoheadrightarrow_{\Sigma'} q'$ is used to denote the existence of a path $q \twoheadrightarrow q'$ where all events are from the set $\Sigma' \subseteq \Sigma$. Notation $P(q, q', \Sigma')$ denotes the set of all paths from q to q' in which only events from Σ' are used.

Definition 2. (Reachability). For EFA $G = (L, D, \Sigma, E, L_0, D_0, L_m)$, a state $q \in Q$ is said to be reachable if $q_0 \twoheadrightarrow q$ for some $q_0 \in Q_0$. The set of reachable states Q^G in G is defined as $Q^G = \{q \in Q \mid \exists_{q_0 \in Q_0} [q_0 \twoheadrightarrow q]\}$.

Definition 3. (Supervisor). Given EFAs G and G', EFA G' is said to be a supervisor of G, if G' is obtained from G, by strengthening guards on transitions in G. This is denoted by $G' \preceq G$. The guard on a transition $e \in E$ is said to be strengthened if for its guard g_e in G and its guard g'_e in G' the following holds: $g'_e \wedge g_e = g'_e$.

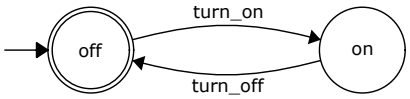
3 PROBLEM DEFINITION

In this section, we first walk through an example to show why the method proposed in [Reijnen et al., 2018] would benefit from an extension to include failure-recovering functionality. We then discuss what properties a failure-recovering supervisor should have and conclude with its formal definition.

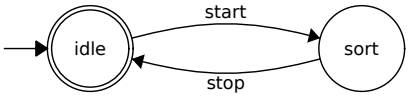
3.1 Example

Suppose a sorting system consists of a conveyor, a sorter, and a diagnoser which detects if the system has jammed. We want to synthesize a fault-tolerant supervisor for this system using the method detailed in [Reijnen et al., 2018]. First, the system components, fault diagnosers, and nominal and post-fault requirements need to be modeled. In [Reijnen et al., 2018], physical relations between components are also modeled, but for simplicity it is assumed that there are no physical relations in the sorting system. To explain the required extension, critical and noncritical states are considered in the context of the synthesized fault-tolerant supervisor.

Components The system has two components: a conveyor and a sorter. Controllable events are used to turn on or start and turn off or stop each of them. In Figure 1, the component models are depicted graphically.



(a) Conveyor



(b) Sorter

Figure 1. The two components of the system.

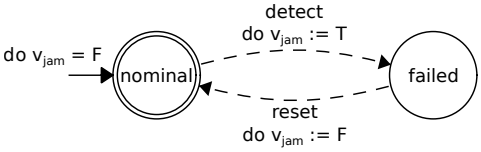


Figure 2. The diagnoser for a jam failure.

Fault diagnoser For the sorting system there is one diagnoser, shown in Figure 2, which diagnoses if a jam failure has occurred in the sorter. The diagnosis of a jam is indicated by variable v_{jam} being true. This might seem unnecessary, given that the diagnosis is also indicated by the states of the diagnoser. However, as mentioned before, this separation is very useful when modeling more complicated diagnoser models.

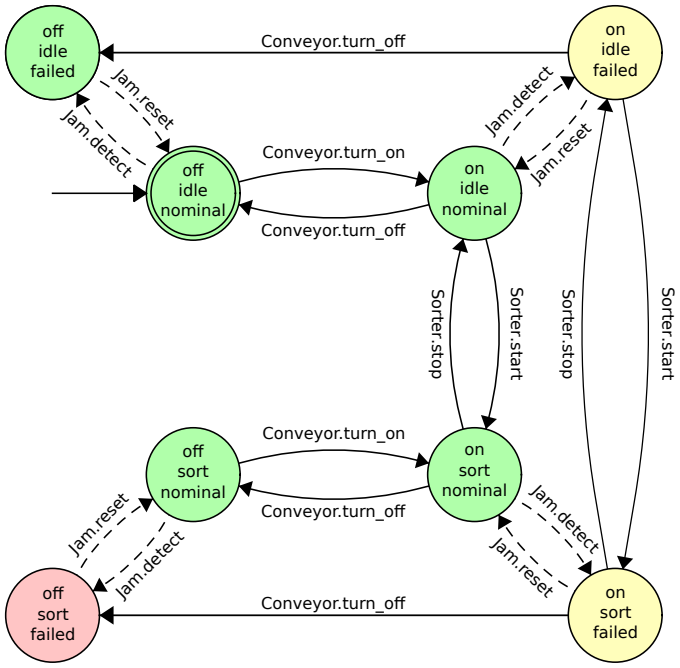


Figure 3. Fault-tolerant supervisor with critical states.

Nominal and post-fault requirements The sorting system has two requirements, which are described with state-based expressions as proposed in [Markovski et al., 2010]. Requirement (R1) states that the sorter events are only allowed when the conveyor is turned on. Requirement (R2) is a post-fault requirement which states that when a jam failure is diagnosed, the conveyor is no longer allowed to turn on again.

(R1) {Sorter.start, Sorter.stop} *needs* Conveyor.on
(R2) Conveyor.turn_on *needs* $v_{jam} \implies F$

Fault-tolerant supervisor Based on the conveyor, sorter and jam diagnoser automata, and the two requirements, the fault-tolerant supervisor is synthesized in the conventional way, of which the FA-equivalent is shown in Figure 3.

In the figure, two classes of states can be distinguished: the inner states in which the system is nominal ($v_{jam} = F$) and the outer states in which the system has failed ($v_{jam} = T$). Intuitively, the nominal states can be considered noncritical. Additionally, all failure states in which the system is regarded as recovered from the failure are noncritical states. Suppose that the above system is regarded to be recovered from the jam failure when both physical components are in their inactive states. Then the state associated with Conveyor.off $\wedge$ Sorter.idle $\wedge$ Jam.failed can be considered noncritical. These noncritical states are colored green.

The remaining failure states in the system can be considered critical. From one of those states, associated with Conveyor.off $\wedge$ Sorter.sort $\wedge$ Jam.failed, no failure recovery using exclusively controllable events is possible. This state is colored red (unrecoverable). The reason why failure recovery should use exclusively controllable events can be seen here. From the red state there is an uncontrollable transition "Jam.reset" to a noncritical state. However, it is not guaranteed that this event will ever occur; the system could be stuck in this state indefinitely. The two yellow states are labelled recoverable, as controllable failure recovery from these states is possible. However, recovery is not guaranteed due to the cycle of controllable events between those states. This can be solved by only allowing transitions which converge to the recovery objective. To conclude, for a system with faults and failures, a supervisor needs to be defined in which critical states are permitted if they are recoverable through controllable events, and in which recovery uses a minimum number of transitions.

3.2 Recovery objective

To ensure that the supervisor can recover from failures, it is important to define which states of the system are noncritical. As illustrated above, the set of noncritical states consists of states in which no failure is present and all failure states which are regarded as recovered. They can be characterized through the fault variables and *recovery objectives*. For each fault in the set of failures $f \in \mathcal{F}_{failure}$, a recovery objective must be defined, which is a predicate $Z_f : D \to \{F, T\}$, which evaluates to true if the system is recovered from the failure. Hence, the set of noncritical states Q_λ for a system with set of states Q is determined as:

$$Q_\lambda = \Big\{(l,d) \in Q \;\Big|\; \forall_{f \in \mathcal{F}_{failure}} \; [v_f(d) \implies Z_f(d)]\Big\}.$$

For the sorting system, the system is regarded to be recovered from the jam failure if both physical components are inactive. Hence, the recovery objective and resulting set of noncritical states for the sorting system are:

$$Z_{jam} = \text{Conveyor.off} \wedge \text{Sorter.idle}$$
$$Q_\lambda = \Big\{(l,d) \in Q \;\Big|\; v_{jam}(d) \implies Z_{jam}(d)\Big\}$$

It can be assumed without loss of generality that the plant and its requirements are represented by the single refined plant EFA $G = (L, D, \Sigma, E, L_0, D_0, L_m)$, set $L_x \subseteq L$ of forbidden locations, and set $Q_\lambda \subseteq Q$ of noncritical states. The set of forbidden states of G is $Q_x = L_x \times D$.

3.3 Failure-recovering supervisor

In this subsection, a supervisor is defined in which all unrecoverable critical states are made unreachable. This supervisor is referred to as the failure-recovering supervisor. In Subsection 3.4, a modification to such a supervisor is introduced to obtain efficient failure recovery, meaning only those paths from a critical to a noncritical state are kept that contain the least number of controllable transitions.

The failure-recovering supervisor is a modification of the proper supervisor described in [Ouedraogo et al., 2011]. A proper supervisor is a supervisor which ensures nonblocking, safety and controllability of the controlled system. Of these properties, *nonblocking* ensures that the system does not end up in a state from which no marked state can be reached. *Safety* ensures that the specified safety requirements are always satisfied. States in which these requirements are not satisfied are forbidden states, which are made unreachable by the supervisor. *Controllability*

ensures that an uncontrollable transition enabled in the plant is not disabled by the supervisor.

For a supervisor to be failure recovering, all its reachable states need to satisfy the *recoverability* property defined below. A state is recoverable if a noncritical state can be reached from it by a path of controllable events. Unrecoverable states should be made unreachable by the supervisor. This way a failure-recovering supervisor is always in "control" of its recovery.

Besides recoverability, a failure-recovering supervisor requires a stricter form of nonblocking. As described in [Moor, 2016] and implemented for fault-tolerant supervisors in [Reijnen et al., 2018], it should not be possible that a marked state can only be reached if a fault or failure occurs. Logically, it is not desirable for the system to have to return to a critical state after just having recovered. Thus, the nonblocking property must be modified, such that a noncritical marked state should always be reachable, without a fault or failure occurring, and without transitioning from a noncritical to a critical state. In the sequel, this property is called FR-nonblocking (nonblocking for failure recovery).

Definition 4. (Recoverability). Given EFA $G = (L, D, \Sigma, E, L_0, D_0, L_m)$ with controllable events $\Sigma_c \subseteq \Sigma$, and given noncritical states Q_λ. State $q \in L \times D$ is recoverable if there is a path $p \in P(q, q_\lambda, \Sigma_c)$ to a noncritical state $q_\lambda \in Q_\lambda$.

Definition 5. (FR-nonblocking). Given EFA $G = (L, D, \Sigma, E, L_0, D_0, L_m)$ with fault events $\Sigma_f \subseteq \Sigma$, and given noncritical states Q_λ. State $q \in L \times D$ is FR-nonblocking if there is a path $p \in P(q, q_{\lambda,m}, \Sigma \setminus \Sigma_f)$ starting in q to a noncritical marked state $q_{\lambda,m} \in Q_\lambda \cap Q_m$, such that for all $1 \leq i < |p|$ it holds that $q_i \in Q_\lambda \implies q_{i+1} \in Q_\lambda$.

Definition 6. (Failure-recovering supervisor). Given EFA G, forbidden locations L_x and noncritical states Q_λ, a recoverable supervisor is defined as a proper supervisor of G with all reachable states satisfying recoverability and FR-nonblocking.

It is preferable that a supervisor only restricts the plant when it is absolutely necessary, otherwise required functionality of the plant could be disabled. A supervisor which disables only the states necessary to satisfy the failure-recovering property is defined to be maximally permissive.

Definition 7. (Maximally permissive failure-recovering supervisor). Let $\mathcal{A}(G)$ be the set of all failure-recovering supervisors of G. A supervisor $G^{s\uparrow} \in \mathcal{A}(G)$ is maximally permissive if $G^s \preceq G^{s\uparrow}$ for any $G^s \in \mathcal{A}(G)$.

3.4 Efficient failure-recovering supervisor

In the failure-recovering supervisor, unrecoverable critical states are made unreachable. However, this supervisor does not guarantee quick and lasting recovery. Firstly, it can contain a critical control loop (an infinite path of critical states connected by controllable events), and never recover to a noncritical state, as in the sorting system example of Figure 3. Secondly, it can contain enabled transitions from noncritical to critical states. For example, if in the sorter system requirement (R2) were not included, after the recovery, it would be possible to immediately turn the conveyor on, and the system would end up in a critical state again. Finally, even if a path to a noncritical state is taken, it might not be the most efficient path.

The supervisor is said to have *efficient failure recovery* if it: (A) enforces efficient failure recovery to a noncritical state using the least possible number of transitions, and (B) enforces that no controllable transitions from a noncritical state to a critical state are possible.

The formal definition of efficient failure-recovering supervisors is given below. First, the failure-recovery distance $\omega^{G^s}(q)$ for a state q is defined.

Definition 8. (Failure-recovery distance). Given EFA G^s, with set of noncritical states Q_λ, the failure-recovery distance for a state $q \in Q^{G^s}$ is defined as the number of transitions in the shortest failure-recovery path from state q (with $\mathcal{T}(p)$ the number of transitions in p):

$$\omega^{G^s}(q) = \min_{q_\lambda \in Q_\lambda} \left[\mathcal{T}(p) \;\middle|\; p \in P(q, q_\lambda, \Sigma_c) \right].$$

Note that for all noncritical states, $q_\lambda \in Q_\lambda$, the failure-recovery distance $\omega^{G^s}(q_\lambda) = 0$. Also note that the failure-recovery distance is infinite if there exists no path to a noncritical state.

Subsequently, using the failure-recovery distance, the efficient failure-recovering supervisor can be defined. In an inefficient failure-recovery path there is at least one pair of adjacent states for which the failure-recovery distance of the first state is not higher than that of the second state. Efficient recovery is accomplished by restricting all controllable transitions to critical states in which the failure-recovery distance does not decrease. This way all inefficient failure-recovery paths are restricted, thus achieving property (A). Since transitions from noncritical to critical states lead to an increasing failure-recovery distance, property (B) is also achieved.

Definition 9. (Efficient failure-recovering supervisor). For a given maximally permissive failure-recovering supervisor G^s and set of noncritical states Q_λ, supervisor $G \preceq G^s$ is efficient if it is failure recovering and each of its edges $(o_e, \sigma_e, g_e, u_e, t_e)$ with a controllable event $\sigma_e \in \Sigma_c$ satisfies the following condition: for all $d \in D$ such that $g_e(d)$ it holds that $(t_e, u(d)) \in Q_\lambda \vee \omega^{G^s}(o_e, d) > \omega^{G^s}(t_e, u(d))$.

Definition 9 suggests a straightforward way of transforming a maximally permissive failure-recovering supervisor into an efficient failure-recovering supervisor by adapting the guards of all edges that do not satisfy the imposed conditions.

In this transformation, only transitions to critical states in G^s are restricted, and per definition a critical state can only be reached after a failure has occurred. All pre-failure transitions are to noncritical states, and remain unaltered. In other words, only post-failure system behavior is altered in the efficient failure-recovering supervisor, and pre-failure system behavior remains unaltered. This is an important observation, because this means that controlled system remains maximally permissive in its nominal behavior.

4 SUPERVISOR DERIVATION

In this section, two algorithms are formulated for the derivation of an efficient failure-recovering supervisor. The first algorithm is used to synthesize a maximally permissive failure-recovering supervisor for an uncontrolled system G. The second algorithm modifies this maximally permissive failure-recovering supervisor such that the recovery is efficient, while the nominal system behavior stays maximally permissive. As a result, the efficient failure-recovering supervisor for G is obtained.

4.1 Synthesis of the maximally permissive failure-recovering supervisor

Algorithm 1 is formulated to compute a maximally permissive failure-recovering supervisor, as defined in Definitions 6 and 7. The algorithm is an adjusted version of the synthesis algorithm introduced in [Ouedraogo et al., 2011]. In [Ouedraogo et al., 2011], a nonblocking predicate N is used, but in Algorithm 1, this is replaced with a robustness predicate R. This predicate is true if a state satisfies FR-nonblocking and recoverability. This algorithm takes as input an EFA G with a set of forbidden locations L_x and a set of noncritical states Q_λ. This algorithm iterates over the set of states in G.

In Algorithms 1 and 2, $o(e)$, $\sigma(e)$, $g(e)$, $u(e)$ and $t(e)$ are defined as respectively the origin location, transition label, guard function, update function and terminal location of transition $e \in E$. Finally, $q_o(e,d)$ and $q_t(e,d)$ are defined as respectively the origin state $(o(e),d)$ and terminal state $(t(e),u(e)(d))$ of transition e for data vector d.

To compute the supervisor SSROB(G,L_x,Q_λ), first the sets of states are initialized on (line 1). Following initialization, the algorithm consists of one outer loop over j, with two inner loops over k and i. The outer loop strengthens the guards of the transitions $e \in E$ iteratively, with the initial guards being the guards of EFA G (line 2).

The first inner loop (lines 5-11) introduces the robustness predicate $R^{j,k}(q)$ which is T if a state q is flagged as FR-nonblocking and recoverable. Initially, this predicate is T if state q is marked and noncritical (line 5). If in an iteration k, state q has an enabled transition $e \in E_q$ to a state which is flagged as robust, then the robustness predicate of q updates to T (line 8). The algorithm keeps updating the predicates, until an iteration of k is reached in which none of the predicates change value.

The second inner loop (lines 12-18) introduces the bad-state predicate $B^{j,i}(q)$ which is T if a state q is flagged as a bad state (a bad state is a state which has to be unreachable in the supervisor). Initially, this predicate is T if state q is a forbidden state, if the robustness predicate of this state is F, or if the bad-state predicate of this state was T in a previous iteration of j (line 12). If a state q has an enabled transition with an uncontrollable event to a state which is flagged as a bad state, then its bad-state predicate updates to T as well (line 15). The algorithm keeps updating the bad-state predicates, until an iteration of i is reached in which none of the bad-state predicates change value.

Algorithm 1 Synthesis of the maximally-permissive failure-recovering supervisor SSROB(G,L_x,Q_λ).

Input: EFA $G = (L, D, \Sigma, E, L_0, D_0, L_m)$, set of forbidden locations $L_x \subseteq L$ and set of noncritical states $Q_\lambda \subseteq L \times D$.

1: Initialize the sets of states: $Q = L \times D$, $Q_m = L_m \times D$, $Q_x = L_x \times D$.

2: Initialize guards: for all $e \in E$ and all $d \in D$, $g_e^0(d) = g(e)(d)$

3: $j = 0$

4: **repeat**

5: Initialize robustness predicates: for all $q \in Q$,

$$R^{j,0}(q) = q \in Q_\lambda \cap Q_m$$

6: $k = 0$

7: **repeat**

8: Update robustness predicates: for all $q = (l_q, d_q) \in Q$,

$$R^{j,k+1}(q) = R^{j,k}(q) \vee \bigvee_{e \in E_q} \left[g_e^j(d_q) \wedge R^{j,k}(q_t(e,d_q)) \right],$$

with $E_q = \Big\{ e \in E \mid o(e) = l_q \wedge \sigma(e) \notin \Sigma_f \wedge (q \in Q_\lambda \Rightarrow q_t(e,d_q) \in Q_\lambda) \wedge (q \notin Q_\lambda \Rightarrow \sigma(e) \in \Sigma_c) \Big\}$

9: $k = k + 1$

10: **until** $\forall_{q \in Q}\ [R^{j,k}(q) = R^{j,k-1}(q)]$

11: for all $q \in Q$, $R^j(q) = R^{j,k}(q)$

12: Initialize bad-state predicates: for all $q \in Q$,

$$B^{j,0}(q) = \begin{cases} T, & \text{if } q \in Q_x \\ \neg R^j(q), & \text{if } q \notin Q_x \text{ and } j = 0 \\ \neg R^j(q) \vee B^{j-1}(q), & \text{if } q \notin Q_x \text{ and } j > 0 \end{cases}$$

13: $i = 0$

14: **repeat**

15: Update bad-state predicates: for all $q = (l_q, d_q) \in Q$,

$$B^{j,i+1}(q) = B^{j,i}(q) \vee \bigvee_{e \in E_q} \left[g_e^j(d_q) \wedge B^{j,i}(q_t(e,d_q)) \right],$$

with $E_q = \Big\{ e \in E \mid o(e) = l_q \wedge \sigma(e) \in \Sigma_u \Big\}$

16: $i = i + 1$

17: **until** $\forall_{q \in Q}\ [B^{j,i}(q) = B^{j,i-1}(q)]$

18: for all $q \in Q$, $B^j(q) = B^{j,i}(q)$

19: Update guards: for all $e \in E$ s.t. $\sigma(e) \in \Sigma_c$, and all $d \in D$,

$$g_e^{j+1}(d) = g_e^j(d) \wedge \neg B^j(q_t(e,d))$$

20: $j = j + 1$

21: **until** $\forall_{e \in E, d \in D}\ [g_e^j(d) = g_e^{j-1}(d)]$

Output: SSROB(G,L_x,Q_λ) $= (L, D, \Sigma, E', L_0, D_0, L_m)$,

$$\text{with } E' = \Big\{ \big(o(e), \sigma(e), g_e^j, u(e), t(e)\big) \;\Big|\; e \in E \Big\}$$

Next, the guards of transitions labelled with a controllable event are strengthened (line 19) with the negation of the bad state predicate. Upon termination of the outer iteration, the algorithm outputs SSROB(G,L_x,Q_λ), the maximally permissive failure-recovering supervisor of G with updated guards.

The only difference between algorithm 1 and the algorithm presented in [Ouedraogo et al., 2011], is that the conditions for updating the robustness predicates, are more strict than those of the nonblocking predicates. Thus, the computational complexity of Algorithm 1 is equal to that of [Ouedraogo et al., 2011], which is $\mathcal{O}(|Q|^2)$.

Theorem 4.1. (Correctness of Algorithm 1). Algorithm 1 terminates and the result SSROB(G,L_x,Q_λ) is the maximally permissive, failure-recovering supervisor of EFA G w.r.t. L_x and Q_λ.

The proof of this theorem can be found in [Paape, 2019].

4.2 Efficient failure-recovering supervisor

Algorithm 2 takes the output of Algorithm 1 and a set of noncritical states Q_λ as input and provides an efficient failure-recovering supervisor, as per Definitions 6 and 9.

Algorithm 2 Calculation of the efficient failure-recovering supervisor SSRCS(G^s,Q_λ).

Input: Recoverable supervisor $G^s = (L, D, \Sigma, E, L_0, D_0, L_m)$, and set of noncritical states $Q_\lambda \subseteq L \times D$.

1: $Q = L \times D$, $Q_m = L_m \times D$
2: $W^0 = Q_\lambda$, $\quad V^0 = Q \setminus W^0$
3: for all $q \in W^0$, $\omega(q) = 0$
4: $i = 0$
5: **repeat**
6: Calculate reachable states:
$W^{i+1} = \left\{ q = (l_q, d_q) \in V^i \;\middle|\; \exists_{e \in E_q}[q_t(e, d_q) \in W^i] \right\}$,
with $E_q = \{e \in E \mid o(e) = l_q \wedge \sigma(e) \in \Sigma_c \wedge g(e)(d_q) = T\}$
7: $V^{i+1} = V^i \setminus W^{i+1}$
8: for all $q \in W^{i+1}$, $\omega(q) = i + 1$
9: $i = i + 1$
10: **until** $W^i = \emptyset$
11: for all $q \in V^i$, $\omega(q) = \infty$
12: Update guards: for all $e \in E$ s.t. $\sigma(e) \in \Sigma_c$, and all $d \in D$,
$g_e^{new}(d) = g(e)(d) \wedge \left[q_t(e, d) \in Q_\lambda \vee \omega(q_o(e, d)) > \omega(q_t(e, d)) \right]$

Output: SSRCS(G^s,Q_λ) $= (L, D, \Sigma, E', L_0, D_0, L_m)$,
with $E' = \left\{ \left(o(e), \sigma(e), g_e^{new}, u(e), t(e) \right) \middle| e \in E \right\}$

First, the failure-recovery distance of every state in Q is calculated iteratively with a breadth-first search (lines 2-10). In this search, two types of sets are used. For an iteration of i, W^i denotes the set of all states for which the failure-recovery distance equals i. V^i denotes the set of all states for which failure-recovery distance $> i$, and thus for which the failure-recovery distance is yet to be calculated. Initially, all noncritical states $q_\lambda \in Q_\lambda$ are in set W^0, and their failure-recovery distance $\omega(q_\lambda) = 0$.

Every iteration, a check is done for every state in V^i (line 6). If a state $q \in V^i$ has an enabled controllable transition to a state which is in the set W^i, then $q \in W^{i+1}$. In (line 6), for all states in set W^{i+1} the failure-recovery distance ω of these states is set to $i + 1$. This value is equal to the number of controllable transitions required to reach a noncritical state. The algorithm keeps determining the failure-recovery distances, until no more states are discovered. In (line 11), for all states q for which there is no failure-recovery path $\omega(q)$ is set to ∞.

Next, on (line 12), the guard of every transition with a controllable event is updated, such that all inefficient failure-recovery paths are disabled. Such a transition is restricted if its terminal state is a critical state, and the failure-recovery distance does not increase from origin state to terminal state. The algorithm terminates by outputting the efficient failure-recovering supervisor SSRCS(G^s,Q_λ).

While Algorithm 1 has two inner and one outer loops, resulting in a complexity of $\mathcal{O}(|Q|^2)$, Algorithm 2 only has one loop, terminating in at most $|Q|$ steps. Thus, if Algorithm 1 can be feasibly executed for a system, then Algorithm 2 is also feasible.

Theorem 4.2. (Correctness of Algorithm 2). Algorithm 2 terminates, and the result SSRCS(G^s,Q_λ) is the efficient failure-recovering supervisor of G^s w.r.t. Q_λ.

The proof of this theorem can be found in [Paape, 2019].

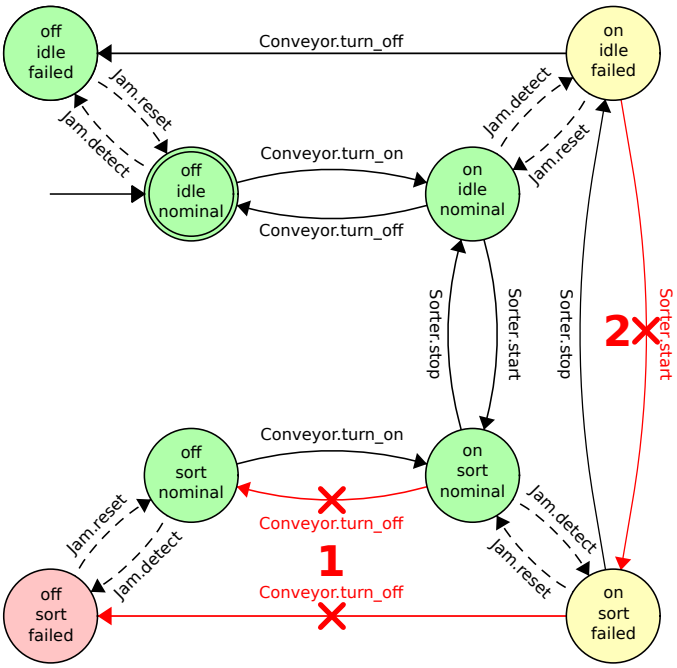


Figure 4. The efficient failure-recovering supervisor of the sorting system is derived through Algorithms 1 and 2.

4.3 Example

Figure 4 shows how a supervisor can be derived for the sorting system using Algorithms 1 and 2. In Algorithm 1, the unrecoverable state associated with Conveyor.off $\wedge$ Sorter.sort $\wedge$ Jam.failed is labelled as nonrobust because it is not recoverable (in red). As a consequence, it is labelled as a bad state, and so is the state associated with Conveyor.off $\wedge$ Sorter.sort $\wedge$ Jam.nominal, which has an uncontrollable transition going to the nonrobust state. Next, the guards of the transitions to these bad states are set to false (the red transitions indicated by a 1). In the following iteration, no guards are changed and the supervisor outputs the failure-recovering supervisor SSROB(G,L_x,Q_λ).

Next, SSROB(G,L_x,Q_λ) is used as input for Algorithm 2. First, the recovery distance is calculated for all states. There is one enabled transition in which the failure-recovery distance in the terminal state is higher than that of the origin state (the red transition indicated by a 2). The guard on this transition is set to false, and the algorithm outputs the efficient failure-recovering supervisor SSRCS(G^s,Q_λ), which is shown in Figure 5.

If we compare the efficient failure-recovering supervisor in Figure 5, to the fault-tolerant supervisor in Figure 3, then we can see that all unrecoverable critical states are made unreachable, and that recovery is always efficient (with the least number of transitions possible). The supervisor is also maximally permissive in its nominal behavior.

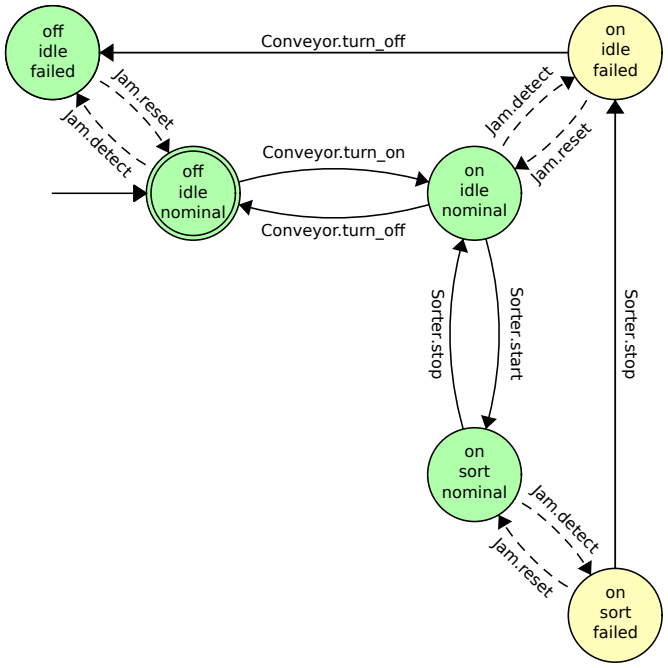


Figure 5. The resulting efficient failure-recovering supervisor of the sorting system.

5 CONCLUDING REMARKS

In this paper, a supervisor is defined with efficient failure recovery. To derive such a supervisor, two algorithms have been developed: one for the synthesis of the maximally permissive failure-recovering supervisor, and one for the modification of this supervisor to obtain efficient failure recovery. This efficient failure-recovering supervisor is maximally permissive in the nominal system behavior. The proposed method allows for adding faults and failures in a modular and intuitive way.

The advantage over fault-tolerant supervisory control approaches without failure recovery is that failures can be dealt with, without placing unwanted and unnecessary restrictions on the nominal system behavior. The advantages over other failure-recovery approaches are threefold. Firstly, failures can be added modularly and at later steps in the design cycle. Secondly, the controlled system can recover quickly from multiple simultaneous failures. Finally, this method is easy and intuitive to use; besides the modelling steps as described in [Reijnen et al., 2018], the only extra step required to synthesize a supervisor is to define the recovery objective for every failure.

As a final note, failure recovery — as defined in this paper — is limited to controllable events, as it is assumed it cannot be guaranteed that uncontrollable events will ever occur. However, this assumption might be more strict than necessary; some uncontrollable events can reasonably be guaranteed to occur. As future work, it might be interesting to partition the event set into progressive and non-progressive events, such as done in [Ware and Malik, 2014], with events which are guaranteed to happen being labelled progressive. Using such a method, a failure-recovering supervisor could be synthesized in which recovery can include uncontrollable events as long as they are progressive.

REFERENCES

Acar, A.N. and Schmidt, K.W. (2015). Discrete event supervisor design and application for manufacturing systems with arbitrary faults and repairs. *IEEE International Conference on Automation Science and Engineering*, 825–830.

Andersson, K., Lennartson, B., Falkman, P., and Fabian, M. (2011). Generation of restart states for manufacturing cell controllers. *Control Engineering Practice*, 19(9), 1014–1022.

Cho, K. and Lim, J. (1998). Synthesis of fault-tolerant supervisor for automated manufacturing systems: a case study on photolithographic process. *IEEE Transactions on Robotics and Automation*, 14(2), 348–351.

Fritz, R. and Zhang, P. (2018). Overview of fault-tolerant control methods for discrete event systems. *IFAC-PapersOnLine*, 51(24), 88–95.

Gertler, J. (2014). Fault Detection and Diagnosis. *Encyclopedia of Systems and Control*, 1–7.

Markovski, J., van Beek, D.A., Theunissen, R.J.M., Jacobs, K.G.M., and Rooda, J.E. (2010). A state-based framework for supervisory control synthesis and verification. In *49th IEEE Conference on Decision and Control*, 3481–3486. IEEE.

Moor, T. (2016). A discussion of fault-tolerant supervisory control in terms of formal languages. *Annual Reviews in Control*, 41, 159–169.

Ouedraogo, L., Kumar, R., Malik, R., and Akesson, K. (2011). Nonblocking and Safe Control of Discrete-Event Systems Modeled as Extended Finite Automata. *IEEE Transactions on Automation Science and Engineering*, 8(3), 560–569.

Paape, N. (2019). *Model-Based Design of a Supervisory Controller for a System with Faults and Failures*. Master's thesis, Eindhoven University of Technology.

Reijnen, F.F.H., Reniers, M.A., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2018). Structured Synthesis of Fault-Tolerant Supervisory Controllers. *IFAC-PapersOnLine*, 51(24), 894–901.

Schuh, M. and Lunze, J. (2016). Fault-tolerant control for deterministic discrete event systems with measurable state. In *American Control Conference*, 7516–7522.

Sköldstam, M., Åkesson, K., and Fabian, M. (2007). Modeling of Discrete Event Systems using Finite Automata With Variables. In *46th IEEE Conference on Decision and Control*, 3387–3392.

Ware, S. and Malik, R. (2014). Progressive events in supervisory control and compositional verification. *Control Theory and Technology*, 12(3), 317–329.

Wen, Q., Kumar, R., and Huang, J. (2014). Framework for optimal fault-tolerant control synthesis: Maximize prefault while minimize post-fault behaviors. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(8), 1056–1066.

Wonham, W.M. and Cai, K. (2019). *Supervisory Control of Discrete-Event Systems*. Communications and Control Engineering. Springer International Publishing.

Zaytoon, J. and Lafortune, S. (2013). Overview of fault diagnosis methods for Discrete Event Systems. *Annual Reviews in Control*, 37, 308–320.

Zaytoon, J. and Riera, B. (2017). Synthesis and implementation of logic controllers – A review. *Annual Reviews in Control*, 43, 152–168.