Model-based design and synthesis of fault-tolerant and failure-recovering supervisors

Paape, N.

Award date:
2019

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mechanical Engineering
Control Systems Technology Research Group

Model-based design and synthesis of fault-tolerant and failure-recovering supervisors

Master Thesis

N. Paape

Supervisors:
TU/e Thesis Supervisor M.A. Reniers
TU/e Supervisor J.M. van de Mortel-Fronczak
External Supervisor L. Swartjes

CST Report 2019.070

Eindhoven, September 2019

Acknowledgements

I like to thank Asia van de Mortel-Fronczak for her great help and advice during these past year, and her willingness to spend many many hours sifting through my many many drafts looking for errors and mistakes. I would also like to thank Lennart Swartjes for his great advice and insights. Whenever I was stuck with a problem or developed a new theory, he would come up with observations which I would never have thought of myself. I think not many students are as lucky as I was to have two supervisors who have as much knowledge between the two of them, and I would not have been able to do this without them. Next, I would like to thank the people at Vanderlande, who gave me this opportunity, and who I greatly enjoyed spending my time with the past year. I could not have wished for a better environment to work in. Finally, I would like to thank everyone else at the Eindhoven University of Technology, who helped me with brainstorming and with critical views on my ideas.

Model-based design and synthesis of fault-tolerant and failure-recovering supervisors

N. Paape, *n.paape@student.tue.nl*
Control Systems Technology Research Group
Department of Mechanical Engineering
Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract

Over the years many methodologies have been developed for fault-tolerant and failure-recovering supervisory control of discrete-event systems. However, due to a lack of a structured supervisor design method, the design of such supervisors is generally complicated. The aim of this paper is to bridge the gap between theory and application, and to do so, a model-based design method for fault-tolerant and failure-recovering supervisors is proposed. The proposed design method features synthesis of the supervisor based on plant and requirement models, which is achieved through two newly developed algorithms. In this design method, special focus is put on clarity, modularity and making modelling more intuitive, which is accomplished through structured design steps featuring state-based expressions, decomposition of the plant and requirements models.

Keywords

Discrete-event systems, supervisory control, model-based engineering, supervisor synthesis, fault-tolerance, failure-recovery

1 INTRODUCTION

In the field of automated logistic systems, Vanderlande is a major player providing logistic system, such as automated baggage handling, parcel sortation and warehousing solutions. These logistic systems require controllers which guarantee the machine safety and liveness (functionality) of the system, even when the system suffers from defects. However, in recent years the complexity of these systems has been increasing rapidly. This leads to more complex machine safety and liveness specifications, and a higher number of possible defects in the system. 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 system's functionality.

The level of control which guarantees machine safety and liveness in a *plant* (uncontrolled system) is the *supervisory control* layer. As mentioned in [1] and [2], a logistic system and its supervisory controller 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 liveness specifications by enabling or disabling controllable events (which the supervisor can prevent from occurring, typically actuator events). Examples of machine safety and liveness specifications in a logistic system are respectively: no packages are released to a conveyor which is full, and each package is eventually released to the conveyor.

The design of a supervisory controller for simple logistic systems can done by hand, but a more streamlined design method and process is warranted for more complex systems. A process which has seen widespread use in industry, and which is also in use at Vanderlande, is the *V-model* [3]. The V-model is a non-linear design process originally proposed to streamline software development, which can also be applied to supervisory control. The V-model is nonlinear in the sense that the initial stages of the project definition phase are very broad, getting more granular as time passes. Then follows the implementation phase, and finally, the project testing and integration phase in

This work was supported by Vanderlande Industries B.V., Veghel, The Netherlands.

which granularity decreases again until the project finalizes. The advantage of the V-model is the verification and validation between the project definition phase and the testing and integration phase in every step of the process. However, the V-model does not prescribe how a supervisory controller can be realized from the specifications. Traditionally, the design method for supervisory controllers is using specification documents describing the requirements and the design of the system to realize the supervisory controller. However, a more recent development described in [4] and [5], is supervisory controller design based on plant and supervisory controller automata models. This method is generally known as the *model-based design* method. It allows for testing and verification of the plant and supervisor before they are (fully) realized. In [3], it is shown that model-based design can be applied effectively for realizing supervisory control in a logistic (baggage handling) system.

System defects have a detrimental influence on the machine safety and liveness of the controlled system [6], and maintaining machine safety and liveness during defects is one of the major challenges in the design of a supervisor. The design methods as described in [4] and [5] are focused on the design of a supervisory controller which guarantees machine safety and liveness during *nominal* (free of defects) plant behaviour, but in practical applications there will be defects, and the supervisory controller must be resilient to them. The two types of defects which will be dealt with in this paper are faults and failures, which are classified in [7] and [8]. A *fault* is a malfunction in the plant which degrades system performance. A faulty plant has different and machine safety and liveness specifications and requires *fault-tolerant control*, a topic which will be discussed in Subsection 2.2. A *failure* is a subclass of faults in which the malfunction results in a total system breakdown; the system becomes unable to fulfil its functionality and depending on what state the system is in, it could be in a critical condition. If the system is in a critical condition, then it needs to return to a noncritical system state as quickly as possible. A supervisor which guarantees that after a failure the system returns to a noncritical system state is a *failure-recovering* supervisor, and work related to such supervisors will be discussed in Subsection 2.3. To the best of the author's knowledge, there is no structured model-based design method for the design of a fault-tolerant and failure-recovering supervisor.

These challenges in supervisory controller design are not limited to the logistic systems of Vanderlande, but hold for a wide range of discrete-event systems [6]. The main goal of this paper is to propose a structured and intuitive design method for the design of fault-tolerant and failure-recovering supervisors for discrete-event systems. A secondary goal is that the supervisor disturbs the plant as little as possible during its nominal behaviour, preferably only when required to fulfil machine safety and liveness specifications. This secondary goal does not apply to defective system behaviour as quick failure-recovery is prioritized over functionality.

1.1 Contribution

To accomplish the above goal, three contributions are made. The first contribution is a formal definition of a fault-tolerant and failure-recovering supervisor, which disturbs the plant as little as possible. The second contribution is the realization of two algorithms for the synthesis of such a supervisor. The third contribution is a structured design method for fault-tolerant and failure-recovering supervisors. In this design method, special focus is put on clarity, modularity and intuitiveness.

1.2 Overview of the paper

This paper is structured as follows. First is an overview of related work in Section 2. Next, are preliminaries in Section 3. The fault-tolerant and failure-recovering supervisor, and two algorithms for the synthesis of such a supervisor are defined in Section 4. The structured design method is introduced in Section 5. Finally, concluding remarks are discussed in Section 6.

2 Related work

An overview is given below of work related to the design of a fault-tolerant and failure-recovering supervisor for discrete-event systems. First an overview is given of design methods, followed by an overview of fault handling in supervisory control. Lastly, an overview is given how a supervisor can be designed which can deal with failures.

2.1 Work related to design methods

In traditional design methods, the design of a system with supervisory control starts with a definition of requirements the controlled system has to fulfil. From these requirements then follows the design

of the controlled system. The design of the controlled system is then used to specify requirements for the plant components and the supervisor components. Derived from these requirements, are the designs for those components. The designs are then realized, and the realized plant and supervisor components are tested with respect to their requirements. Subsequently, the realized components are integrated together to form the controlled system. Finally, the controlled system is verified with respect to the initial requirements. If the controlled system does not fulfil its initially specified requirements, then requirements and designs need to be redefined and redesigned till it does.

In the model-based design method, component models are created following the component designs. This allows for testing of the designs before the components are realized. The upsides of this design method are that the system can be simulated and verified before being built and that it allows for early integration; realized components can be tested with yet-to-be-realized components. It is also easier to adapt the design of a component when discrepancies are found between the performance of the designed system and required system performance. In [9] many approaches to model-based design are discussed in the context of supervisory control, mentioning IBM Telelogic Harmony-SE, INCOSE Object-Oriented Systems Engineering Method (OOSEM), IBM Rational Unified Process for Systems Engineering (RUP SE) for Model-Driven Systems Development (MDSD) and Vitech Model-Based Systems Engineering (MBSE) Methodology. However, as stated in [9], these methodologies are focused on design model formulation and not on derivation of a supervisor design.

A formal method which can be used to simplify the design of supervisors for discrete-event systems is *supervisory control theory* (SCT). A design approach which incorporates SCT in model-based design, is shown in [9] and [10]. This extension of model-based design allows for the supervisor to be *synthesized* (generated) from a model of the plant and a model of the controlled system requirements which the supervisor has to fulfil. The model-based design method with supervisory control synthesis is shown in Figure 1. The advantages of supervisory control synthesis are speeding up the controller development cycle and guaranteeing that machine safety and liveness specifications are implemented correctly; verification of the supervisor with respect to its requirements becomes unnecessary. Synthesis also allows for properties such as safety, controllability and nonblocking to be guaranteed by the supervisor (these properties are further explained in Section 4). These advantages can be clearly seen in applications of model-based design with supervisory control synthesis, examples of which are a driver assistance system [11], a patient support system [12] and a theme park vehicle [5].

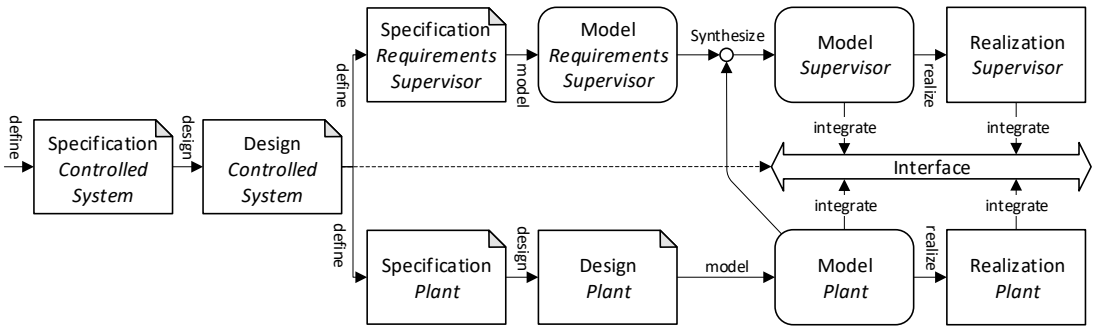


Figure 1: The model-based design method with supervisory control synthesis [9].

2.2 Work related to fault-tolerant control

The model-based design method with supervisory control synthesis as proposed in [9] and [10], is focused on nominal working systems, and does not describe how resilience to faults and failures can be guaranteed. The first step in making a system resilient is to implement *fault diagnosis* which, as the name suggests, is diagnosing if a fault has occurred in the plant. As shown in [13] and Figure 2, there are three types of faults: permanent, drift-like and intermittent. Examples of these are respectively: a component which breaks, a component which gradually wears down over time, and a component with a bad wiring contact. In this paper only permanent and intermittent faults are considered, as drift-like faults are hard to deal with as they do not occur at a discrete instance in time. They are generally dealt with by lower-level controllers [14].

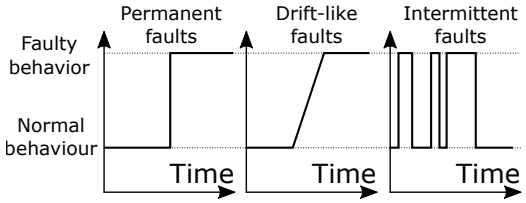


Figure 2: Fault types, from [13].

After diagnosis of a fault, the plant is determined to be no longer operating in a nominal state, but in a *post-fault* state. When the plant is in a post-fault state, its behaviour might be different, and the same goes for the required machine safety and liveness specifications. The result is that the supervisor designed for a nominally working system does not necessarily guarantee functionality and safety for a system that is in a post-fault state. *Fault-tolerant control* (FTC) for SCT is needed to guarantee machine safety and liveness specifications whether the system is in a nominal or in a post-fault state [15].

Approaches to FTC are generally divided into two categories: passive and active, as discussed in [16]. A passive *fault-tolerant control system* (FTCS) is a single controller that handles both the nominal and post-fault system. The downsides of a passive FTCS are that more restrictions on the nominal system might be imposed than necessary, and that it is not guaranteed that one controller exists which satisfies all specified requirements for both nominal and post-fault system states. One approach of passive FTC is introduced in [17], in which it is regarded as a supervisory control problem under partial observation. In this approach, faults do not need to be diagnosed, as the supervisor which is synthesized can accommodate both nominal and post-fault system behaviour. However, if certain behaviour is not allowed in the post-fault system, then it might be unnecessarily restricted in the nominal system.

The other type of FTC is active FTC, an overview of which is given in [6]. For an active FTCS there is a supervisor which imposes the required nominal behaviour, and only after a fault has been diagnosed the supervisor is adjusted accordingly; the controlled system uses different controllers for nominal and post-fault states of the system. The main downside of an active FTCS is that the fault has to be diagnosed before the controller can be adjusted. During the diagnosis period the nominal controller will be used on the post-fault system, and the system might be unsafe during this duration.

For a design method which should work on a wide range of systems, an active FTCS is preferable. It is more flexible, and it is preferable that nominal system behaviour is as permissive as possible, which cannot be guaranteed when a passive FTCS is used. In [14], a structured FTC design method is proposed, in which an active FTCS is synthesized using guards in extended finite-state automata (EFA) and by describing system specifications with state-based expressions as proposed in [18]. This method offers the following advantages: it uses structured modelling steps, the state-based requirements intuitively follow from supervisor requirement specifications, it can handle multiple faults occurring simultaneously, and it supports the modular additions of faults. These advantages allow for a good integration into the model-based design paradigm.

2.3 Work related to failure-recovery control

To create a supervisor which is resilient to failures, it is important to first narrow down how machine safety can be guaranteed during those failures. As stated before, a failure is seen as a defect in the system after which it is potentially in an unsafe state. There are two types of states the system can be in when it comes to failures. It can be in a *noncritical state*, which is a state in which either no failure has been diagnosed, or a failure has been diagnosed but the system is deemed not dangerous. The opposite of a noncritical state is a *critical state*, in which a failure has been diagnosed and the system is in a dangerous state.

The supervisor has no control over when a failure occurs, so the system can unexpectedly enter a critical state if a failure occurs while carrying out its required functionality. If a supervisor is designed such that the system is never allowed to be in a critical state then, as a result, required functionality of the system might need to be disabled. The resulting supervisor would be very restrictive, as shown later in this paper in the example in Section 5.

Safety in SCT is generally guaranteed by preventing unsafe system states to be reached. However, as mentioned above, it is not possible to prevent critical states from being reached without restricting the system's required functionality. Still, allowing the system to be in a critical state does not satisfy

machine safety. To create a supervisor which satisfies machine safety, while still allowing for its intended functionality, another approach has to be taken. Critical states are instead permitted temporarily. However, it is required that the system recovers quickly to a noncritical state.

Multiple approaches to failure-recovery already exist. One such approach is introduced in [19]. In that paper, a framework is proposed for the design of a supervisory controller, in which the post-failure system recovers to a pre-failure system state. However, this would not work in a system with permanent failures, e.g. when a component breaks down permanently. A similar approach of failure recovery is proposed in [20], in which recovery is not to a pre-failure system state, but to a recovered post-failure system state as specified by the designer. This is done by defining nominal system specifications, degraded post-failure specifications, and recovered post-failure specifications.

Failure-recovery can also be guaranteed through reactive synthesis. Reactive supervisor synthesis, such as in [21] and [22], is different from supervisory control synthesis. In reactive supervisor synthesis, requirements can be specified with modal logic, e.g. a recovery event must be enabled after a failure event. A case has been used to test the reactive synthesis approach proposed in [21], and the outcome was that reactive synthesis can be used to synthesize a supervisor with failure-recovery. However, it also demonsted that this approach suffers from the state-space explosion problem for complex systems, as a supervisor could only be synthesized for systems with low complexity.

The issue with all of the above approaches is that recovery is expressed to happen eventually, but the system can stay in a critical state for an arbitrarily long time. For failures it is critical that a recovery is reached as quickly as possible, and not eventually. A technique which can be applied to guarantee a recovery is executed quickly is that of event enforcement, which involves the notation that enforced events can pre-empt other events. By introducing forceable events, a recovery can be forced, such that the system does not stay in a critical state indefinitely. A few approaches such as event enforcement by adaptation of the formal framework, or event enforcement by adjustment of the modelling process are discussed in [23]. The main issues with event enforcement is that only controllable events can be enforced by the supervisor, and an event cannot actually be guaranteed to pre-empt uncontrollable events in a physical system. This makes reaching a solution for failure recovery not as simple as implementing event enforcement in the above methodologies.

For a supervisor to be able to recover as quickly as possible, it needs to always be in control of its recovery. This means that if the supervisor is in a critical state, then a noncritical state needs to be reachable through a *path* of controllable events (a path is a number of events in succession). If this condition is satisfied then the critical state is said to be *recoverable*. Figure 3 shows which parts of the state-space should be permitted and which should be disallowed (note: not all possible scenarios are included in the figure). Post-failure states are only allowed if they are recoverable (there is a path of controllable events to a noncritical state), and any state from which an unrecoverable state can be reached through a path of uncontrollable events should be disallowed.

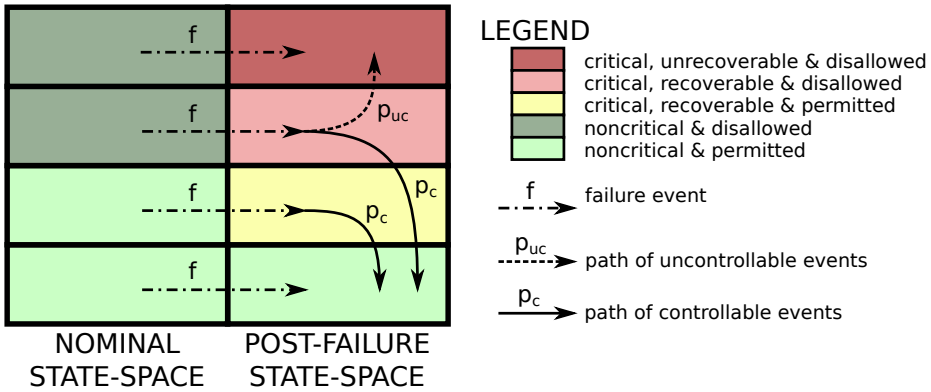


Figure 3: The nominal and post-failure state-space.

Finally, quick recovery to a noncritical state must be enforced by the supervisor. A quick recovery is achieved by enforcing *optimal-recovery*. What this means is that the system enforces the path with the least number of controllable events to a noncritical state. This enforcement can be achieved by disabling all non-optimal recovery paths. To the author's knowledge there is no definition yet for a supervisor which guarantees that all failures are recoverable, and which guarantees optimal-recovery. Neither is there a method to synthesize a supervisor which satisfies these properties.

3 PRELIMINARIES

The goal of this paper is to propose a structured design method for fault-tolerant and failure-recovering supervisors for discrete-event systems. However, to accomplish this goal, it is first important to define how such a system can be modelled, what a fault-tolerant and failure-recovering supervisor is, and how one can be synthesized. To do so, some preliminaries are provided.

In Subsection 2.1, model-based design with supervisory control synthesis as described in [9] was shown to be an effective design method for a multitude of applications. However, this design method does not allow for the design of a supervisor which is resilient when it comes to faults and failures. Fault-tolerant control as described in [14] can be used for a design method which synthesizes a fault-tolerant supervisor. However, the supervisor synthesis algorithm needs to be adjusted to create a supervisor which is resilient both to faults and failures, instead of just faults. In [14], modelling is not done with standard finite-state automata (FA), but with extended finite-state automata (EFA), and the synthesis is based on the synthesis algorithm defined in [24].

3.1 EFAs

An FA has locations and transitions between those locations. An EFA is an FA extended with variables, guards and updates. An EFA is defined as a 7-tuple $A = (L, D, \Sigma, E, L_0, D_0, L_m)$. In EFAs, transitions are extended with guards, which are predicates denoting for which variable values the transition is enabled, and updates, which are functions which assign new values to the variables after the transition. In this paper, all variables must have finite domain, for example booleans and integers with a finite range.

The finite set of locations is L, with $l \in L$ denoting one location. $L_0 \subseteq L$ is the set of initial locations (there can be more than one initial location, the initial location does not need to be predetermined), and $L_m \subseteq L$ is the set of marked locations.

When modelling the system, a finite number of variables are used, all with finite domains. The set of finite possible combinations of variable values, with p variables in the EFA, is described by data set $D = D^1 \times \cdots \times D^i \times \cdots \times D^p$, with D^i representing the domain of one variable. The values of all variables in D at any given moment are represented by a vector of length p, and is denoted by $d \in D$ which is defined as $d = [d(1), \ldots, d(i), \ldots, d(p)]$, with one variable defined as $d(i) \in D^i$. The initial set of data values $D_0 \subseteq D$ is $D_0 = D_0^1 \times \cdots \times D_0^i \times \cdots \times D_0^p$ (similarly to locations, there can be more than one initial value).

Σ is the finite set of events, and $\sigma \in \Sigma$ denotes an event in this set. The set of events is partitioned into two types: controllable and uncontrollable, which a supervisor respectively can and cannot disable. $\Sigma_c \subseteq \Sigma$ is the set of controllable events and $\Sigma_u = \Sigma - \Sigma_c$ is the set of uncontrollable events. The set of fault events is a subset of the set of uncontrollable events $\Sigma_f \subseteq \Sigma_u$, and the set of failure events is a subset of that $\Sigma_{failure} \subseteq \Sigma_f$.

The finite set of transitions is $E \subseteq L \times \Sigma \times \mathcal{G} \times \mathcal{F} \times L$, with $\mathcal{G}$ the set of guard predicates and $\mathcal{F}$ 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 $e = (o_e, \sigma_e, g_e, f_e, t_e)$ is a 5-tuple, as defined below. A transition e is enabled when the system is in location o_e and when $g_e(d) = T$.

- $o_e \in L$ 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.
- $f_e : D \to D$ is the data update function of e.
- $t_e \in L$ is the terminal location of e.

3.2 Graphical representation

An EFA can be represented graphically, which is done as follows. A location is depicted by a circle, and a marked state is depicted by a double circle. An initial location has an unconnected arrow pointing towards the location, and the initial values of all variables are defined for this initial location. A solid or dashed arrow respectively depicts a transitions with an controllable or uncontrollable event. Transitions are labelled by: "σ when g_e do f_e", with σ the event, g_e the guard predicate, and f_e the update function. The guard and the update are optional and can be omitted.

To give an example, below is the EFA of a counter, which counts till 3. The counter has one location l_1, which is both the initial and the marked location, so: $L = L_m = L_0 = \{l_1\}$. The counter has

one variable: $d(1) = d^{\text{count}}$. This variable is an integer with a range of $[0, ..., 3]$, so the domain of $d(1)$ is: $D^1 = \{0, ..., 3\}$. The initial value of d^{count} is 0, so $D^0 = \{0\}$. The counter has one transition with controllable event "c_increase", which is a self-loop in location l_1 (a self-loop is a transition which originates and terminates in the same location). The transition of the self-loop is described as $e_1 = (l_1,\ \sigma_1,\ [d(1) < 3],\ [d(1) := d(1) + 1],\ l_1)$.

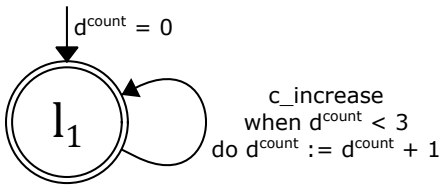


Figure 4: The EFA of a counter.

3.3 FA equivalence of EFA

Because an EFA has a finite domain of finitely bounded variables, each EFA can be transformed to an equivalent FA, as shown in [25]. Thus, supervisor properties such as nonblocking and controllability (see Section 4) which are defined for FAs, can also be defined for EFAs. When trying to define properties for EFAs, it is useful to first have insight in how the FA equivalent of an EFA can be constructed.

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 $Q = L \times D$. An initial state is defined as $q_0 = (l_{q_0}, d_{q_0})$ with $l_{q_0} \in L_0$ and $d_{q_0} \in D_0$, with the set of initial states $Q_0 = L_0 \times D_0$. A marked state is defined as $q_m = (l_{q_m}, d_{q_m})$ with $l_{q_m} \in L_m$ and $d_{q_m} \in D$, with the set of marked states $Q_m = L_m \times D$. It is not necessarily true that every state in sets Q or Q_m can be reached from an initial state.

Besides states, an EFA transition $e \in E$ can be represented in an equivalent FA as follows. The origin state of a transition is $q = (l_q, d_q)$, with $l_q = o_e$. If the transition is enabled then $g_e(d_q) = T$. Finally, the terminal state of a transition from state $q \in Q$ is the combination of the transition its terminal location and the updated variables values $\bar{q}_{e,d_q} = (t_e, f_e(d_q))$. If transition e from state q is said to be *restricted*, then the guard of the transition is made stronger such that $g_e(d_q) = F$.

As an example, the FA equivalent to the counter in Figure 4 is shown below in Figure 5, in which all states are combinations of the location $l1$ and the values of variables d^{count}. The example shows that a transition from one location can correspond to the same transition from multiple states. Most supervisor properties apply to states and the transitions between those states, and thus in the rest of this paper states will be used to define these properties where possible.

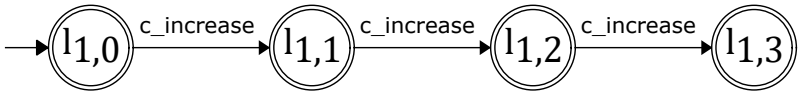


Figure 5: The FA equivalent of the EFA shown in Figure 4.

3.4 System modelling

When synthesizing the supervisor for a system, the system is commonly split into a plant model and a requirement model. Just as in [14], the plant is modelled by decomposing it into component models and models for the physical relations between those components (the reason for modelling physical relations is given in Subsection 5.3). This makes modelling the plant more intuitive and increases modularity. For synthesis, the models for these components and physical relations first need to be unified to the combined plant. For EFAs, the combined plant can be constructed with the parallel composition of all z plants: $P = P_1||...||P_z$, as described in [24].

Similarly to [14], predicates are used with location-based variables. A location-based variable is a finite enumeration variable for which the values correspond to locations in a plant. When a transition terminates in a location, the value of the location-based variable is updated to correspond to the specific location the plant is in. Location-based variables function the same as regular variables, except that the variable initialization and updates are implicit and do not need to be modelled explicitly.

After modelling of the plant, safety and liveness requirements need to be modelled. In this paper these requirements are described with state-based expressions as proposed in [18], as state-based requirements are generally more intuitive to use for designers than requirements based on automata. State-based requirements are in the form of σ needs Y, meaning that event $\sigma \in \Sigma$ is only enabled when predicate Y is satisfied. These predicates can consist of regular variables and location-based variables. How the requirement model R is obtained from the plant model and the state-based expressions is explained in [18].

For synthesis of the supervisor to be possible, the plant model P must first be refined with respect to the modelled safety and liveness requirements R. What refinement means is that a singular "refined" plant EFA $G = (L, D, \Sigma, E, L_0, D_0, L_m)$ with forbidden locations L_x is obtained. Refined plant G has the same behaviour as P, but in G all locations which do not satisfy the requirements R are in the set of forbidden locations L_x. The refined set of locations of plant G is obtained as following: $L = L^P \times (L^R \cup \{\phi\})$, and the set of forbidden locations $L_x = L^p \times \phi$, with ϕ the terminal location of all transitions which are enabled in P, but not in R. How the other parameters of G are determined is explained in Section III of [24]. The set of forbidden states of G is $Q_x = L_x \times D$.

3.5 Faults and failures

The set of all faults in a system is denoted by $\mathcal{F}$. The set of all failures is a subset of the set of all faults: $\mathcal{F}_{failure} \subseteq \mathcal{F}$. Each fault $f \in \mathcal{F}$ has a corresponding boolean fault variable d^f, the value of which is true if fault f has been diagnosed.

To ensure that the supervisor can recover from failures, it is important to define which states of the system are noncritical states. The set of noncritical states is all nominal states, and all states which are deemed to be noncritical post-failure. The first is described with the fault-variable, and the latter can be intuitively be described through what will be defined as *recovery-objectives*. For a failure $f' \in \mathcal{F}_{failure}$, a recovery-objective is a predicate $Z_{f'}$ which needs to be fulfilled for the system to be regarded as recovered, with $Z_f : D \to \{F, T\}$.

This means that for failure f', the statement which describes if the system is in a noncritical condition, is $(d^{f'} \implies Z_{f'})$. Suppose that in state $q \in Q$, the value of fault variable $d^{f'}$ is denoted by $d_q^{f'}$, and the evaluation of predicate $Z_{f'}$ is denoted by $Z_{f'}(d_q)$. Then, the set of noncritical states Q_λ for a system with set of states Q can be determined as:

$$Q_\lambda = \left\{ q \in Q \middle| (\forall f' \in \mathcal{F}_{failure})[d_q^{f'} \implies Z_{f'}(d_q)] \right\}$$

To give a short example, imagine there is machine with two locations: on and off. The machine has one failure f_{broken}, which is denoted through variable d^{broken}. Suppose for this failure, the recover-objective is that the machine should be off. Then Z_{broken} = (Machine.off) and $Q_\lambda = \{q \in Q | (d_q^{broken} \implies Z_{broken})\}$. So Q_λ is all nominal states, and all post-failure states for which the machine is in the off location.

From here onwards it is assumed without loss of generality that the entire plant and its requirements are represented by the singular refined plant EFA $G = (L, D, \Sigma, E, L_0, D_0, L_m)$, and the set of forbidden locations $L_x \subseteq L$, and a set of noncritical states $Q_\lambda \subseteq Q$. To define a fault-tolerant failure-recovering supervisor for this system, the following three definitions first need to be given: paths, reachability and subautomata. From here onwards when referring to an automaton, an extended finite-state automaton is intended.

Definition 1 (Path). *Given EFA G, by $q_i \xrightarrow{\sigma_i} q_{i+1}$ we denote that a transition $e_i \in E$ from state $q_i = (l_{q_i}, d_{q_i})$ to $q_{i+1} = (l_{q_{i+1}}, d_{q_{i+1}})$ such that: $e_i = (l_{q_i}, \sigma_i, g_{e_i} : g_{e_i}(d_{q_i}) = T, f_{e_i} : f_{e_i}(d_{q_i}) = d_{q_{i+1}}, l_{q_{i+1}})$. A path in G between two states with $q_1 = q = (l_q, d_q)$ and $q_{n+1} = q' = (l_{q'}, d_{q'})$, with all states in the path in Q, is denoted by:*

$$p^G_{q \to q'} = q_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{r-1}} q_r \xrightarrow{\sigma_r} \cdots \xrightarrow{\sigma_n} q_{n+1}$$

For a path, the notation $p^G_{q \to q'}$ can be used to describe a path in G. A path using only events from a subset of events $\Sigma' \subseteq \Sigma$ is denoted as $p^G_{q \xrightarrow{\Sigma'} q'}$. The number of states in a path $p^G_{q \to q'}$ is denoted by $|p^G_{q \to q'}|$.

Definition 2 (Reachability). *Given EFA G, a state $q \in Q$ is reachable if there exists a path $p^G_{q_0 \to q}$ from an initial state $q_0 \in Q_0$ to q. Similarly, a state $q' \in Q$ is said to be reachable from state q if there is a path $p^G_{q \to q'}$. The set of reachable states Q^G in G is defined as $Q^G = \{q \in Q | \exists p^G_{q_0 \to q} \; s.t \; (q_0 \in Q_0)\}$.*

Definition 3 (Subautomaton). *Given EFA G, automaton G' is said to be a subautomaton of G, if G' is obtained from G, by making guards on transitions in G stronger. This is denoted by $G' \preceq G$.*

4 The Fault-tolerant and Failure-recovering Supervisor

The goal of this paper is to propose a structured and intuitive design method for the design of a fault-tolerant and failure-recovering supervisor. To accomplish this, it will first be defined what the properties of a fault-tolerant and failure-recovering supervisor are, and how one can be synthesized.

A proper supervisor as described in [24], is a subautomaton of G, that 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 supervisor 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. However, for machine safety and liveness to be satisfied in a system with faults and failures, the supervisor needs to satisfy additional properties besides safety, nonblocking and controllability.

First, in Subsection 4.1, a supervisor is defined which is a fault-tolerant and failure-recoverable supervisor (recoverable, not recovering!), which will be referred to as a *recoverable supervisor*. However, as mentioned in Subsection 2.3, there needs to be a mechanism through which quick recovery is enforced. In Subsection 4.1 a recoverable supervisor with optimal-recovery is defined, with failure-recovery being optimal if the least number of controllable transitions are taken to a noncritical state. For both of these supervisors a synthesis algorithm is formulated. The fault-tolerant and failure-recovering supervisor for G is obtained by first synthesizing the recoverable supervisor G^s, and then applying optimal-recovery synthesis to G^s.

4.1 The recoverable supervisor

To define recoverable supervisor G^s, we first define the concept of failure-recoverability. In Subsection 2.3, it was shown that critical states cannot always be forbidden without disabling required functionality of the system. Instead, critical states are permitted, as long as the controlled system is always able to recover to a noncritical state. An issue with recovery was that of event enforcement. Firstly, only controllable events can be enforced by the supervisor [23], and secondly, uncontrollable events cannot always be pre-empted. So for a supervisor G^s to be *failure-recoverable*, there needs to be a path from every state $q \in Q^{G^s}$ to a noncritical state, using only controllable events. If a supervisor satisfies failure-recoverability, then the supervisor is always in "control" of its recovery.

When defining a supervisor which is resilient to faults and failures, there are is another condition which needs to be satisfied besides failure-recoverability. In [15], it is shown that a fault should never become inevitable. In other words, for every reachable state $q \in Q^{G^s}$ a path to a marked state $q_m \in Q_m$ must exist, such that for the path:

(I) There are no transitions from noncritical to critical states.
(II) No transition uses a fault event.
(III) The marked state q_m is in the set of noncritical states Q_λ.

A state is defined to satisfy *fault-evitability* if from the state a path to a marked state exists which satisfies properties (I), (II) and (III). Figure 6 give examples of states which, from top to bottom, does not satisfy (I), does not satisfy (II), does not satisfy (III), and finally, satisfies all three.

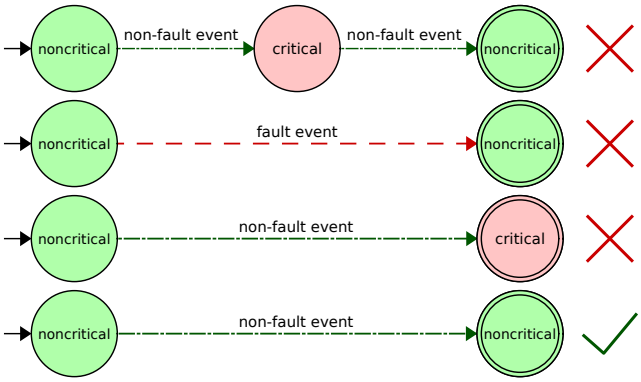


Figure 6: Examples of states which do not (first three) and do satisfy fault-evitability (last).

A supervisor which satisfies nonblocking, safety, controllability, failure-recoverability and fault-evitability is defined to be a recoverable supervisor, for which the formal definition is given in Definition 4.

Definition 4 (Recoverable supervisor)**.** *Given automaton $G = (L, D, \Sigma, E, L_0, D_0, L_m)$, forbidden locations L_x and noncritical states Q_λ. The set of states of G is $Q = L \times D$. A recoverable supervisor G^s is defined as a subautomaton of G for which all reachable states $q \in Q^{G^s}$ are nonblocking, safe, controllable and failure-recoverable and fault-evitable.*

Nonblocking *State q is nonblocking if there exists a path starting in q, terminating in a marked state $q_m \in Q_m$. So if in state q the following property is satisfied: $\exists p^{G^s}_{q \to q_m}$ s.t. $\big(q_m \in Q_m\big)$.*

Safety *State q is safe if it is not a forbidden state, with set of forbidden states $Q_x = L_x \times D$, so: $q \notin Q_x$.*

Controllability *State q is controllable if it is safe and if all states to which q has a transition with an uncontrollable event are not removed in the supervisor G^s. The set of states reached by a transition from $q \in Q^G$ with an uncontrollable event in the set Σ_u is defined as: $Q^G_{q,\Sigma_u} = \{q' \in Q^G | \exists q \xrightarrow{\sigma} q' \text{ s.t. } (\sigma \in \Sigma_u)\}$. So q is controllable if it is safe and if $Q^G_{q,\Sigma_u} \subseteq Q^{G^s}$.*

Failure-recoverability *State q is failure-recoverable there if is a path starting in q to a noncritical state $q_\lambda \in Q_\lambda$, such that: every transition is controllable and the path terminates as soon as a noncritical state is reached. This condition is defined formally below. For a path $p^{G^s}_{q \to q_\lambda}$, state q_i and e_i are defined as the ith state and transition in the path respectively. Note that this condition is satisfied if $q \in Q_\lambda$.*

$$\exists p^{G^s}_{q \to q_\lambda} \text{ s.t. } (q_\lambda \in Q_\lambda) \wedge \Big(\forall i : 1 \leq i < |p^{G^s}_{q \to q_\lambda}|, (\sigma_{e_i} \in \Sigma_c) \wedge (q_i \notin Q_\lambda)\Big)$$

Fault-evitability *State q is fault-evitable if there is a path starting in q to a marked, noncritical state $q_{m,\lambda} \in (Q_\lambda \cap Q_m)$, such that: there are no transitions from a noncritical to critical state, and there are no transitions with fault events. This condition is defined formally below. For a path $p^{G^s}_{q \to q_{m,\lambda}}$, state q_i and e_i are defined as the ith state and transition in the path respectively. Note that this condition is satisfied if $q \in (Q_\lambda \cap Q_m)$.*

$$\exists p^{G^s}_{q \to q_{m,\lambda}} \text{ s.t. } \big(q_{m,\lambda} \in (Q_\lambda \cap Q_m)\big) \wedge \Big(\forall i : 1 \leq i < |p^{G^s}_{q \to q_{m,\lambda}}|, (\sigma_{e_i} \notin \Sigma_f) \wedge (q_i \in Q_\lambda \implies q_{i+1} \in Q_\lambda)\Big)$$

It is preferable that the supervisor does not disturb the plant when its not necessary, otherwise required functionality of the plant could be disabled. To accomplish this it is required that the supervisor does not disable more states than absolutely necessary. A supervisor which disables only the states necessary to satisfy the recoverable supervisor properties is defined as the maximally permissive recoverable supervisor.

Definition 5 (Maximally permissive recoverable supervisor)**.** *A maximally permissive recoverable supervisor $G^{s\uparrow}$ is defined as the supervisor which respectively disables and restricts only the necessary states and transitions of G to satisfy the recoverable supervisor properties. Suppose $\mathcal{A}(G)$ is the set of all possible recoverable supervisors of G. All possible recoverable supervisors are subautomata of the maximally permissive recoverable supervisor: $G^s \in \mathcal{A}(G),\ \ G^s \preceq G^{s\uparrow}$.*

To obtain the maximally permissive recoverable supervisor of G, Algorithm 1 is conceived. The algorithm yields EFA SSROB(G), the maximally permissive recoverable supervisor of G.

4.1.1 Supervisory synthesis of the maximally permissive recoverable supervisor

Algorithm 1 is formulated to compute a maximally permissive recoverable supervisor, as defined in Definitions 4 and 5. The algorithm is an adjusted version of the synthesis algorithm introduced in [24]. In [24], 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 nonblockingness, failure-recoverability, and fault-evitability. 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. First, two remarks are made below which are needed to explain how predicates are updated in the algorithm.

Remark 1. *Suppose that $\forall i \in \mathbb{N}_0$, we have predicates Y^i and Z^i with $Y^{i+1} = Y^i \vee Z^i$.*
If for an $(x \in \mathbb{N}_0)(i = x), Y^i = T$, then: $(\forall i : i \geq x), Y^i = T$.

Remark 2. *Suppose that $\forall i \in \mathbb{N}_0$, we have predicates Y^i and Z^i with $Y^{i+1} = Y^i \wedge Z^i$.*
If for an $(x \in \mathbb{N}_0)(i = x), Y^i = F$, then: $(\forall i : i \geq x), Y^i = F$.

To compute the supervisor SSROB(G), 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 first outer loop over j strengthens the guards of the transitions $e \in E$, with the initial guards being the guards of EFA G (line 2).

The first inner loop over k (lines 5-11) introduces the robustness predicate $R^{j,k}(q)$ which is T if a state q is flagged as nonblocking, failure-recoverable, and fault-evitable. Initially this predicate is T if state q is marked and noncritical (line 5). If on 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). By applying Remark 1 we can show that once a predicate is updated to T, then it stays T for further iterations of k. The algorithm keeps updating the predicates, until an iteration of k is reached in which none of the predicates change value, after which $R^j(q) = R^{j,k}(q)$ on (line 11), with $R^j(q) = T$ for all states which satisfy nonblockingness, failure-recoverability, and fault-evitability.

The second inner loop over i (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 on 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). By applying Remark 1 we can show that once a predicate is updated to T, then it stays T for further iterations of i. 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, after which $B^j(q) = B^{j,i}(q)$ on (line 18), with $B^j(q) = T$ for all states which should be made unreachable by the supervisor due to nonblockingness, controllability, safety, failure-recoverability, or fault-evitability not being satisfied.

Next, the guards of transitions are updated on (line 19). If for iteration j the terminal state of a transition with a controllable event has bad-state predicates which is T, then that guard is disabled. If not, or if the event is uncontrollable, then that guard remains unchanged. By applying Remark 2 we can say that once a guard is updated to F, then it stays F for further iterations of j. If at least one guard changes value, then another iteration of j is carried out (line 21). If not, then the loop over j terminates, the new event guards are determined on (line 22), and the algorithm outputs SSROB(G), the maximally permissive recoverable supervisor of G.

4.1.2 Correctness of Algorithm 1

For Algorithm 1 one proposition and two theorems are provided, stating the correctness of the algorithm. Proposition 1 states that the algorithm terminates. Theorem 4.1 states that the supervisor SSROB(G) resulting from Algorithm 1 is nonblocking, safe, controllable and failure-recoverable and fault-evitable. Theorem 4.2 states that SSROB(G) is the maximally permissive recoverable supervisor of G. Proofs for the proposition and theorems are provided in Appendix A.

Proposition 1 (Termination of Algorithm 1)**.** *Given EFA G with forbidden locations L_x and noncritical states Q_λ, the computation of SSROB(G) terminates in a finite number of steps.*

Theorem 4.1 (Nonblockingness, safety, controllability, failure-recoverability and fault-evitability of SSROB(G))**.** *Given EFA G with forbidden locations L_x and noncritical states Q_λ, SSROB(G) is nonblocking, safe, controllable, failure-recoverable and fault-evitable as long as none of its initial states is a bad-state.*

Theorem 4.2 (Maximal permissiveness of SSROB(G))**.** *Given EFA G with forbidden locations L_x and noncritical states Q_λ, SSROB(G) is the maximally permissive recoverable supervisor of G.*

Algorithm 1 Supervisory synthesis for a recoverable supervisor SSROB(G).

Input: EFA $G = (L, D, \Sigma, E, L_0, D_0, L_m)$ with 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: $Q = L \times D$. The set of marked states is $Q_m = L_m \times D$, and the set of forbidden states is $Q_x = L_x \times D$. A state $q \in Q$ is a combination of its location and data value $q = (l_q, d_q)$. The origin state $q_{e,d} \in Q$ of a transition $e \in E$ from a state with data value $d \in D$ is $q_{e,d} = (o_e, d)$. The terminal state of a transition $e \in E$ from a state with data value $d \in D$ is $\bar{q}_{e,d} = (t_e, f_e(d))$, and from a state $q \in Q$ with data value $d_q \in D$ the terminal state is $\bar{q}_{e,d_q} = (t_e, f_e(d_q))$.

2: Initial guards: $\forall e \in E, \forall d \in D : g_e^0(d) = g_e(d)$

3: Initialize iterator $j := 0$

4: **repeat**

5: Initialize the robustness predicates:

$$\forall q \in Q, \qquad R^{j,0}(q) = \begin{cases} T, & \text{if } q \in (Q_m \cap Q_\lambda); \\ F, & \text{if } q \notin (Q_m \cap Q_\lambda). \end{cases} \tag{1}$$

6: Initialize iterator $k := 0$

7: **repeat**

8: Update the robustness predicates:

$$\forall q \in Q, \qquad R^{j,k+1}(q) = R^{j,k}(q) \vee \bigvee_{e \in E_q} \left[g_e^j(d_q) \wedge R^{j,k}(\bar{q}_{e,d_q}) \right], \tag{2}$$

$$\text{with } E_q = \left\{ e \in E | (o_e = l_q) \wedge (\sigma_e \notin \Sigma_f) \wedge (q \in Q_\lambda \implies \bar{q}_{e,d_q} \in Q_\lambda) \wedge (q \notin Q_\lambda \implies \sigma_e \in \Sigma_c) \right\}.$$

9: $k := k + 1$

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

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

12: Initialize the bad-state predicates:

$$\forall q \in Q, \qquad B^{j,0}(q) = \begin{cases} T, & \text{if } q \in Q_x; \\ \neg R^j(q), & \text{if } q \notin Q_x \text{ and } j = 0; \\ \neg R^j(q) \vee B^{j-1}(q), & \text{if } q \notin Q_x \text{ and } j > 0. \end{cases} \tag{3}$$

13: Initialize iterator $i := 0$

14: **repeat**

15: Update the bad-state predicates:

$$\forall q \in Q, \qquad B^{j,i+1}(q) = B^{j,i}(q) \vee \bigvee_{\{e \in E | o_e = l_q \wedge \sigma_e \in \Sigma_u\}} \left[g_e^j(d_q) \wedge B^{j,i}(\bar{q}_{e,d_q}) \right]. \tag{4}$$

16: $i := i + 1$

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

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

19: Update the guards:

$$\forall e \in E, \forall d \in D, \qquad g_e^{j+1}(d) = \begin{cases} g_e^j(d) \wedge \neg B^j(\bar{q}_{e,d}), & \text{if } \sigma_e \in \Sigma_c; \\ g_e^j(d), & \text{if } \sigma_e \in \Sigma_u. \end{cases} \tag{5}$$

20: $j := j + 1$

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

22: $\forall e \in E, g_e(d) = g_e^j(d)$

Output: SSROB(G) $= (L, D, \Sigma, E, L_0, D_0, L_m)$ with $E = \{(o_e, \sigma_e, g_e, f_e, t_e) | \forall e \in E\}$.

4.2 The optimally-recovering supervisor

In the algorithm for synthesis of the recoverable supervisor, unrecoverable states are made unreachable. Hover, the recoverable supervisor does not enforce recovery to a noncritical state. Machine safety is not guaranteed in a recoverable supervisor for a few reasons. Firstly, the recoverable supervisor could be in a control loop (an infinitely long path of controllable events), and never recover to a noncritical state. Secondly, the supervisor can contain enabled transitions from noncritical to critical states. Lastly, even if a path to a noncritical state is taken, it might not be the most optimal path (which is the path with the least number of transitions). A supervisor needs to be defined which: (A) enforces optimal failure-recovery to a noncritical state, and (B) enforces that no controllable transitions from a noncritical state to a critical state are possible. If both of these conditions are satisfied, then the supervisor is said to be *optimally-recovering*. A supervisor which is safe, controllable, nonblocking, failure-recoverable, fault-evitable and optimally-recovering, is defined as a fault-tolerant and failure-recovering supervisor. The formal definition of an optimally-recovering supervisor is given in this subsection. But first, failure-recovery distance $\omega^{G^s}(q)$ for a state q is defined.

Definition 6 (Failure-recovery distance)**.** *Given EFA G^s, with the 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 (and is infinity if there is no such path):*

$$\omega^{G^s}(q) = \min\left\{\left|p^{G^s}_{q\rightarrow q_\lambda}\right| \text{ s.t. } (q_\lambda \in Q_\lambda) \wedge \left(\forall i : 1 \leq i < |p^{G^s}_{q\rightarrow q_\lambda}|, (\sigma_{e_i} \in \Sigma_c) \wedge (q_i \notin Q_\lambda)\right)\right\},$$

with state q_i and e_i defined as the ith state and transition in path $p^{G^s}_{q\rightarrow q_\lambda}$ respectively.

The value of $\omega^{G^s}(q)$ denotes the number of transitions in the shortest possible failure-recovery path in state q in supervisor G^s. For all noncritical states, $q_\lambda \in Q_\lambda$ the failure-recovery distance $\omega^{G^s}(q_\lambda) = 0$. For the shortest failure-recovery path from state q, going backwards from the last state which is a noncritical state, the failure-recovery distance increases by one for every transition until state q is reached. The failure-recovery distance for q is $\omega^{G^s}(q) = r$ with r transitions in the shortest failure-recovery path.

For the same state q, the number of transitions in a every non-optimal recovery path is greater than the failure-recovery distance $\omega^{G^s}(q)$. This means there is at least one transition in every non-optimal failure-recovery path, in which the failure-recovery distance of its origin state is not higher than that of its terminal state. Optimal-recovery is accomplished by restricting all controllable transitions to critical states in which the failure-recovery distance does not decrease. By doing so, for (A) all non-optimal failure-recovery paths are restricted. For (B), transitions from noncritical to critical states lead to an increasing failure-recovery distance, and thus these too are restricted. With the above solution, the optimally-recovering supervisor G^{rcs} of G^s can be defined.

Definition 7 (Optimally-recovering supervisor)**.** *Given recoverable supervisor G^s and noncritical states Q_λ. Let G^{rcs} denote the optimally-recovering supervisor of G^s. Supervisor G^{rcs} is a subautomaton of G^s such that non-optimal failure-recovery paths, and transitions from noncritical to critical states are restricted. The optimally-recovering supervisor G^{rcs} is obtained by exclusively restricting transitions in states of G^s for which the following conditions are met: the failure-recovery distance increases, the transition event is controllable, and the terminal state is not a noncritical state. For $d \in D$, transition $e \in E$ has origin state $q_{e,d} \in Q^{G^s}$ and terminal state $\bar{q}_{e,d} \in Q^{G^s}$. So formally, $g_e(d)$ should be F in G^{rcs} if the following condition is satisfied:*

$$(\bar{q}_{e,d_q} \notin Q_\lambda) \;\wedge\; (\omega^{G^s}(q) \leq \omega^{G^s}(\bar{q}_{e,d_q})) \wedge (\sigma_e \in \Sigma_c)$$

Figure 7 shows an example of how an optimally-recovering supervisor is obtained. In the example green states are noncritical state, and red states are critical states. All transitions are controllable, except the failure transition which is uncontrollable. In every state, its failure-recovery distance is depicted, with 0 for failure-safe states. Suppose that in the recoverable supervisor, all transitions are enabled. Transition A is a transition which is in a non-optimal recovery path (the failure-recovery distance does not decrease). Transition B is a transition from a noncritical state to a critical state. Then, to obtain the optimally-recovering supervisor, transitions A and B need to be restricted and the other transitions remain unaltered.

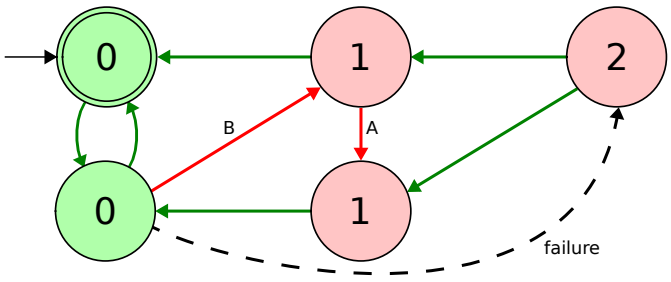


Figure 7: An example of how an optimally-recovering supervisor is obtained.

Theorem 4.3 states that if G^s is a recoverable supervisor of G, then G^{rcs}, the optimally-recovering supervisor of G^s, is also a recoverable supervisor of G. This means that G^{rcs} is the fault-tolerant and failure-recovering supervisor of G. The proof for this is given in Appendix B.1.

Theorem 4.3 (Safety, controllability, nonblockingness, fault-evitability and failure-recoverability of G^{rcs})**.** *Given recoverable supervisor G^s, and optimally-recovering supervisor G^{rcs}. If G^s is safe, controllable, nonblocking, fault-evitable and failure-recoverable, then G^{rcs} is also safe, controllable, nonblocking, fault-evitable and failure-recoverable.*

If G^s is the maximally permissive recoverable supervisor of G, and the fault-tolerant and failure-recovering supervisor G^{rcs} is obtained by restricting transitions in G^s, then G^{rcs} is not a maximally permissive supervisor of G unless $G^{rcs} = G^s$. However, to construct G^{rcs} from G^s, 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 behaviour is altered in the optimally-recovering supervisor, and pre-failure system behaviour remains unaltered. To give an example of this, in Figure 7, transitions between nominal states are left unaltered (the two states on the left). This is an important observation, because this means that nominal behaviour in G^{rcs} is as permissive as it is in the maximally permissive recoverable supervisor G^s; the optimally-recovering supervisor does not disrupt the plant more than absolute necessary to fulfil the safety, controllability, nonblocking, failure-recoverability and fault-evitability properties. Post-failure behaviour is not as permissive, but this is acceptable as machine safety is more crucial than maintaining functionality after a failure.

4.2.1 Supervisory synthesis of the optimally-recovering supervisor

Previously, Algorithm 1 was introduced for the synthesis of a maximally permissive recoverable supervisor. In this subsection Algorithm 2 is introduced. This algorithm takes the output of Algorithm 1, SSROB(G) as input, and synthesizes an optimally-recovering supervisor, as per Definitions 4 and 7.

First, the failure-recovery distance of every state in Q is calculated iteratively over i with a breadth-first search on (lines 4-10), 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. Initializations for the search are first done on (lines 4-5). All noncritical states $q_\lambda \in Q_\lambda$ are in set W^0, and their failure-recovery distance $\omega(q_\lambda) = 0$.

Every iteration of i a check is done for every state in Q on (line 7). If a state $q \in V^i$ has an enabled controllable transition to a terminal state which is in the set W^i, then $q \in W^{i+1}$. Subsequently, V^{i+1} denotes the set of all states for which failure-recovery distance $> i + 1$. On (line 7) for all states in set W^{i+1} the failure-recovery distance ω of these states is determined to be equal 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 an iteration of i is reached in which no new states are found and $W^i = \emptyset$ on (line 10). On (line 11), for all states q for which there is no failure-recovery path we have $\omega(q) = \infty$.

Next, on (line 12), the guards of every transition is updated, such that all non-optimal failure-recovery paths are disabled. So a transition is restricted if it uses a controllable event, if its terminal state is not a noncritical state, and if the failure-recovery distance decreases from origin state to terminal state. Finally, the algorithm outputs the optimally-recovering supervisor SSRCS(G), which is the fault-tolerant and failure-recovering supervisor of G and then terminates.

4.2.2 Correctness of Algorithm 2

For Algorithm 2 a proposition and theorem are provided, stating the correctness of the algorithm. Proposition 2 states that the algorithm terminates in a finite number of steps. Theorem 4.4 states that the supervisor SSRCS(G) resulting from Algorithm 1 is the optimally-recovering supervisor of G^s. If G^s is a recoverable supervisor of G, then Theorem 4.3 indicates that SSRCS(G) is the fault-tolerant and failure-recovering supervisor of G. Proofs for these are provided in Appendix B.

Proposition 2 (Termination of Algorithm 2)**.** *Given EFA G^s with noncritical states Q_λ, the computation of SSRCS(G) terminates in a finite number of steps.*

Theorem 4.4 (SSRCS(G) is the optimally-recovering supervisor of SSROB(G))**.** *Given EFA G^s with noncritical states Q_λ, SSRCS(G) is the optimally-recovering supervisor of G^s.*

Algorithm 2 Supervisory synthesis for a optimally-recovering supervisor SSRCS(G).

1: **Input**: Recoverable supervisor $G^s = (L, D, \Sigma, E, L_0, D_0, L_m)$ with set of noncritical states $Q_\lambda \subseteq L \times D$.
2: Initialize the sets of states: $Q = L \times D$. The set of marked states is $Q_m = L_m \times D$, and the set of forbidden states is $Q_x = L_x \times D$. A state $q \in Q$ is a combination of its location and data value $q = (l_q, d_q)$. The origin state $q_{e,d} \in Q$ of a transition $e \in E$ from a state with data value $d \in D$ is $q_{e,d} = (o_e, d)$. The terminal state of a transition $e \in E$ from a state with data value $d \in D$ is $\bar{q}_{e,d} = (t_e, f_e(d))$, and from a state $q \in Q$ with data value $d_q \in D$ the terminal state is $\bar{q}_{e,d_q} = (t_e, f_e(d_q))$.
3: Initialize iterator $i := 0$.
4: Initialize the failure-recovery sets:

$$W^0 = Q_\lambda, \tag{6}$$
$$V^0 = Q \setminus W^0. \tag{7}$$

5: Initialize the failure-recovery distances:

$$\forall q \in W^0, \qquad \omega(q) = 0. \tag{8}$$

6: **repeat**
7: Update the failure-recovery sets:

$$W^{i+1} = \{q \in V^i | \exists e \in E, o_e = l_q, \sigma_e \in \Sigma_c, g_e(d_q) = T, \bar{q}_{e,d_q} \in W^i\}, \tag{9}$$
$$V^{i+1} = V^i \setminus W^{i+1}. \tag{10}$$

8: Calculate the failure-recovery distances:

$$\forall q \in W^{i+1}, \qquad \omega(q) = i + 1. \tag{11}$$

9: *i:=i+1.*
10: **until** $W^i = \emptyset$.
11: Set the failure-recovery distances of all states without a failure-recovery path:

$$\forall q \in V^i, \qquad \omega(q) = \infty. \tag{12}$$

12: Update the guards to ensure optimal-recovery of the supervisor:

$$\forall e \in E, \forall d \in D, \;\; g_e(d) = \begin{cases} g_e(d) \wedge \neg \left[(\bar{q}_{e,d} \notin Q_\lambda) \wedge (\omega(q_{e,d}) \leq \omega(\bar{q}_{e,d})) \right], & \text{if } \sigma_e \in \Sigma_c; \\ g_e(d), & \text{if } \sigma_e \in \Sigma_u. \end{cases} \tag{13}$$

Output: SSRCS(G) $= (L, D, \Sigma, E, L_0, D_0, L_m)$ with $E = \{(o_e, \sigma_e, g_e, f_e, t_e) | \forall e \in E\}$.

5 Design Method

In this section, a design method is proposed for supervisors which are resilient to faults and failures. The design method for a such a supervisor is based on model-based design as described in [9] and [10], extended with structured fault-tolerant control as described in [14], and with fault-tolerant and failure-recovering supervisor synthesis as described in Section 4. In [14], the design method of the fault-tolerant supervisor is split into two segments, first the nominal system is modelled and the associated supervisor is synthesized. Next, the same is done for the system with faults included. However, in this paper some changes are made to the proposed design method.

The first change, is that nominal and post-fault modelling and synthesis are not explicitly separated. Iterative design of the supervisor is recommended, but how this is executed is up to the designer. Separating nominal and post-fault design is supported in this methodology, but if some faults or failures are known, then including those right away will likely lead to a shorter total design time. However, after synthesis of the supervisor, the nominal behaviour of the controlled system should be validated with respect to its designed specifications.

The second change, is that plant modelling is decomposed into three categories; components, diagnosers, and physical relations. This makes it more intuitive and clear what the relations are between the various components and diagnosers, and modelling is easier as physical dependencies between components are often already described in specification documents.

The third change, is that not only physical relations between components are to be modelled, but also the physical relations between diagnosers and components (e.g. a conveyor breakdown will not be diagnosed when a sensor determines that it is off), or between diagnosers and other diagnosers (e.g. if the conveyor breaks down, a domino effect of faults might be able to occur). Some faults will only occur in some particular states of the system. Also, faults might result in different physical relations between diagnosers and components (e.g. if the conveyor's sensor is faulty, the conveyor can break down whether the sensor declares it is on or off), or between diagnosers and other diagnosers (e.g. if

the conveyor has broken down, other faults might no longer be able to occur).

When all components, diagnosers and physical relations are modelled, nominal and post-fault requirements have to be modelled just as in [14]. After these requirements are modelled, the recovery-objectives needs to be defined. Finally, the supervisor is synthesized. First recoverable supervisor synthesis is accomplished with Algorithm 1, followed by optimally-recovering supervisor synthesis with Algorithm 2, resulting in the fault-tolerant and failure-recovering supervisor of the system. Finally, the controlled system is verified with respect to the initial system specifications. To summarize, The proposed design method has the following design steps:

1) Model components.
2) Model fault diagnosers.
3) Model nominal and post-fault physical relations.
4) Model nominal and post-fault requirements.
5) Model recovery-objectives.
6) Synthesize and validate the fault-tolerant and failure-recovering supervisor.
   Return to step 1 if the controlled behaviour does not conform to the system requirements.

Figure 8 shows how the above design steps fit into the model-based design process as proposed in [9] and shown in Figure 1. Steps (1), (2) and (3) belong to the modelling of the plant, steps (4) and (5) belong to modelling of the supervisor requirements, and synthesis consists of consecutive application of Algorithms 1 and 2 to the plant and requirement models.

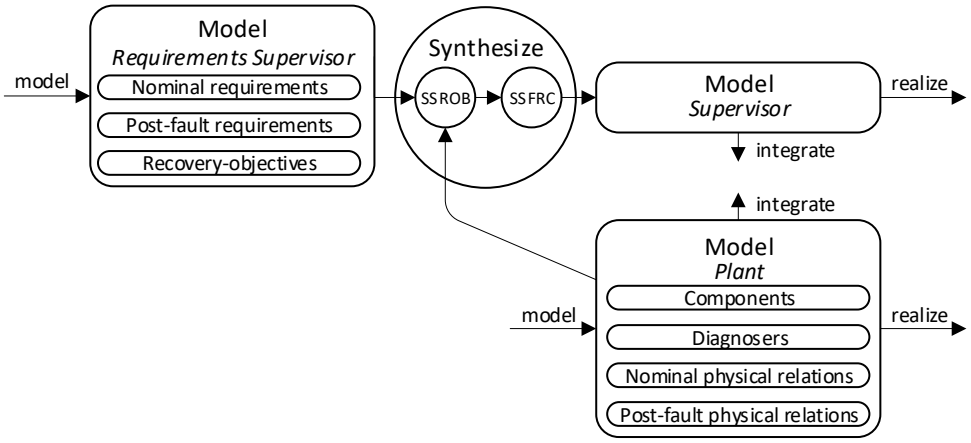


Figure 8: How the design steps fit into the model-based design process as shown in Figure 1.

In the following subsections, the design steps are explained in more detail, and are demonstrated with an example. The example is a system with a conveyor, a sorter and one failure denoting that the sorter has ”jammed”. To keep the example concise, the focus in the example is on clarifying failure-recovery control. In [14] an example is given which better shows the aspects of fault-tolerant control.

5.1 Model components

The first step in the proposed design method is the modelling of the components. Just as in [14], component models are abstractions of their dynamical behaviour. The events in the component models are related to the inputs and outputs relevant to the supervisor. The component models do not change after a fault.

Example:
The sorting system has a conveyor with a sorter. The conveyor can be turned off and on with controllable events, and the sorter can start or stop sorting with controllable events. Below in Figure 9 are the component models for the (a) conveyor and (b) sorter.

(a) Conveyor (b) Sorter

Figure 9: The two components of the system: (a) conveyor, (b) sorter.

5.2 Model fault diagnoser

Diagnosis is essential in making a system resilient to faults and failures. The modeller is free to choose a diagnosis approach most suited to the problem at hand. The only requirements are that:

(A) All faults $f \in \mathcal{F}$ are *diagnosable*, which is, that every fault can be detected within a finite number of events.

(B) For all faults $f \in \mathcal{F}$, an abstraction can be made of its diagnoser, such that there is an EFA with two uncontrollable events denoting detection and resetting of the fault, and with a boolean variable d^f which is true if the fault has been diagnosed.

For every fault $f \in \mathcal{F}$ a diagnoser such as described in (B) must be modelled.

Example:
For the sorting system there is one diagnoser, shown in Figure 10, which diagnoses if the sorter has "jammed". The diagnosing and reset events for this jam are "u_jam_diagnosed" and "u_jam_reset" respectively, and the diagnosis of a jam is indicated by variable d^{jam} being true.

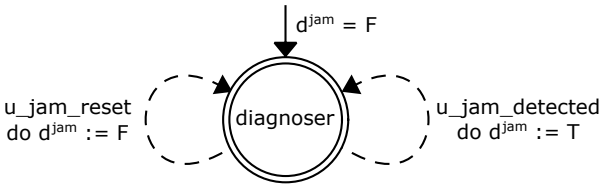
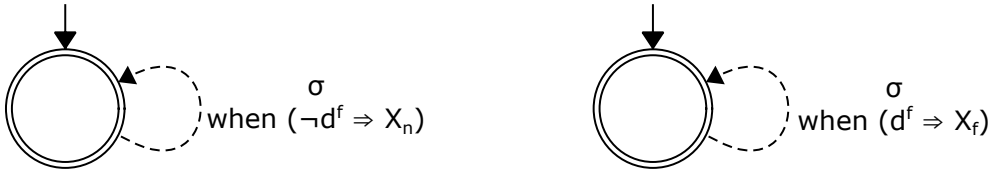


Figure 10: The Diagnoser EFA for a jam failure.

5.3 Model physical relations

As discussed in [26] and formalized in [14], physical relations between components need to be modelled. The first reason for modelling physical relations is to avoid deadlock. A marked state might never be reached, because the uncontrollable transition necessary to reach a marked state cannot physically happen. The second reason is that the supervisor might restrict system behaviour to avoid an uncontrollable transition to a deadlock state, even if the uncontrollable transition cannot physically happen, which means the supervisor is more restrictive than needed. The physical relations are modelled using EFAs with self-loops with guards representing the physical relation between the self-loop event and the state the system is in.

These physical relations can be subject to change after faults, and the physical relations for (a) nominal and (b) post-fault are modelled in separate EFAs, as shown in Figure 11. Guards are applied to the self-loops to switch between the nominal and post-fault behaviours. Physical relations can be described with the following expressions: "σ *when* $(\neg d^f \Rightarrow X_n)$" for nominal behaviour, and σ *when* $(d^f \Rightarrow X_f)$" for post-fault behaviour. In these physical relations transitions, σ is the relevant event, d^f is the boolean fault variable, X_n and X_f are respectively the nominal and post-fault physical relation event guards, and $\Rightarrow$ and $\neg$ are the implication and negation operators, respectively. Below, in Figure 11, two EFAs for (a) nominal and (b) post-fault physical behaviour are given. Remember that failures are a subset of faults, and can cause a change in physical behaviour too.



(a) *Nominal behaviour* (b) *Post-fault behaviour*

Figure 11: Automata for (a) nominal and (b) post-fault physical relations.

Example:
The sorting system has only one physical relation, which is between the diagnoser and the sorter. This physical relation is the same for nominal and post-fault behaviour and is shown in Figure 12. This model describes that the jam failure can only occur when the sorter is sorting.

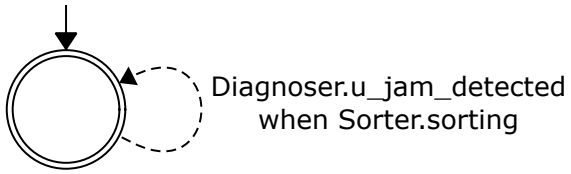


Figure 12: The physical relation between the Diagnoser and Sorter.

5.4 Model nominal and post-fault requirements

Safety and liveness requirements are described with expressions similar to those for physical relations. Nominal requirements are expressed in the form of "σ *needs* $(\neg d^f \Rightarrow Y_n)$" and post-fault behaviour in the form of "σ *needs* $(d^f \Rightarrow Y_f)$", with Y_n and Y_f as the nominal and post-fault requirement event guards respectively.

Example:
Below are the requirements for the example sorting system. Requirement (A) states that the sorter events are only allowed when the conveyor is turned on. Requirement (B) is a post-fault requirement which states that when a jam failure is diagnosed, the conveyor is no longer allowed to turn on again. Requirement (B) could be simplified, however, this is not done to maintain consistency in the notation of requirements.

(A) {Sorter.c_startsorting, Sorter.c_stopsorting} *needs* Conveyor.on
(B) Conveyor.c_on *needs* $d^{jam} \implies F$

5.5 Model recovery-objectives

As discussed in Subsection 3.5, for each failure $f' \in \mathcal{F}_{failure}$, the recovery-objective predicate $Z_{f'}$ is to be defined with a state-based expression. The predicate denotes under which post-failure conditions the system is in a noncritical state, or more colloquially: what state of the system does the designer want the system to recover to after a failure is diagnosed.

$$Q_\lambda = \left\{ q \in Q \middle| \forall f' \in \mathcal{F}_{failure}, [d_q^{f'} \implies Z_{f'}(d_q)] \right\}$$

When defining recovery-objectives, it is important to remember that all transitions in a failure-recovery must be controllable. What this means in practice is that the definition of $Z_{f'}$ must only depend on actuator states and not sensor states. To give an example, imagine our sorting system has a sensor which detects if the conveyor is actually turned on or off. A system designer might assume that after a failure the system must recover to a state in which the sensor is in the off location. However, sensor events are uncontrollable, and a recovery-objective should be something which the supervisor has control over, it cannot enforce what the sensor should detect.

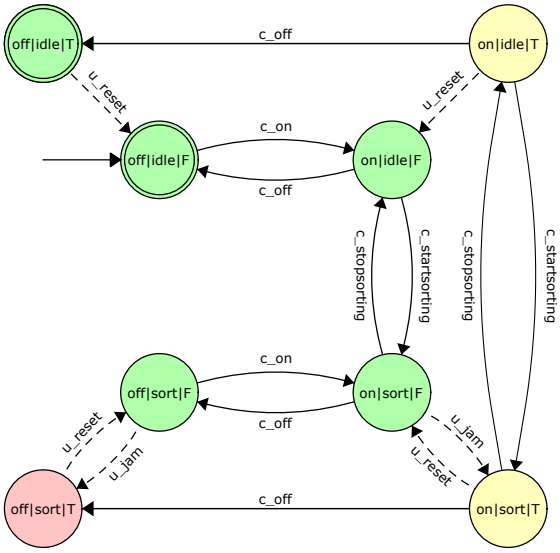
Example:
For the sorting system there is only one failure, denoted by d^{jam}. For the jam failure, the recovery-objective is that the conveyor should be set to off and the sorter should be set to idle. What this means is that if a failure is diagnosed, the system will recover to a state in which the conveyor is off and the sorter is idle as quickly as possible. The recovery-objective predicate for the jam is $Z_{jam} =$ (Conveyor.off $\wedge$ Sorter.idle). The value of d^{jam} in a state q is d_q^{jam}. This results in the following set of noncritical states for this system:

$$Q_\lambda = \{q \in Q^G | (d_q^{jam} \implies (\text{Conveyor.off} \wedge \text{Sorter.idle}))\}$$

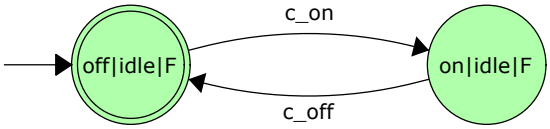
5.6 Synthesis

Finally, the combined plant is to be constructed through parallel composition of the component, diagnoser and physical relation models. The resulting combined plant is then refined with respect to the requirements as described in Subsection 3.4. Next, synthesis is applied to the resulting refined plant. Algorithm 1 is used to first synthesize a maximally permissive recoverable supervisor. Next, Algorithm 2 is used to synthesize the fault-tolerant and failure-recovering supervisor.

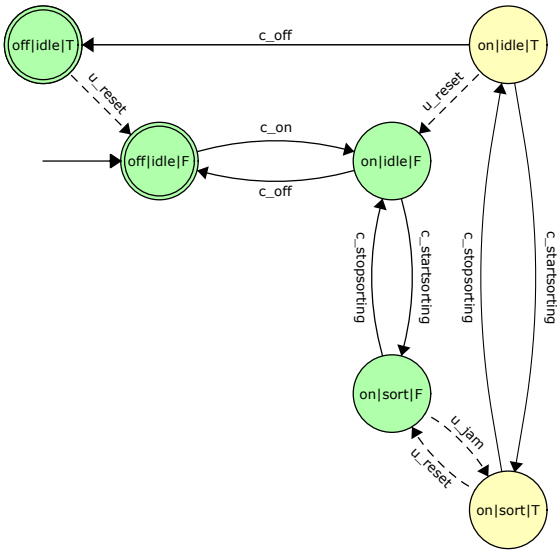
After synthesis, the model of the supervisor and the model of the plant are used to verify the controlled system with respect to the initial system specifications. If the system under control of the supervisor does not fulfil specifications, then the specification and design of the controlled system are respectively redesigned and redefined, and another iteration of the proposed design method is performed. This process is repeated till the controlled system behaves as intended.



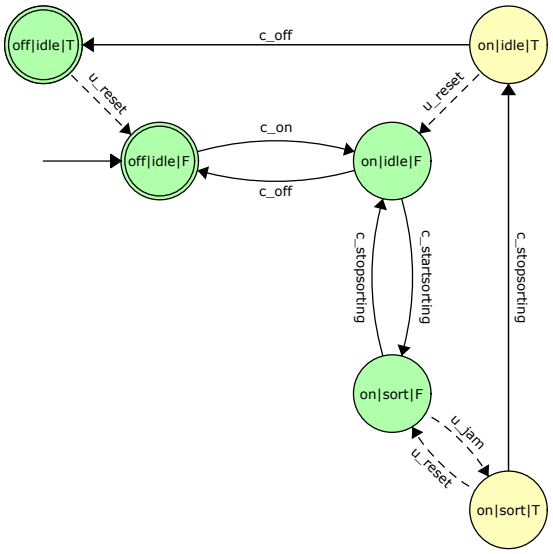
(a) Supervisor SSEFA(G) with critical states allowed.



(b) Supervisor SSEFA(G) with critical states forbidden.



(c) Maximally permissive recoverable supervisor SSROB(G)



(d) Fault-tolerant and failure-recovering supervisor SSRCS(G)

Figure 13: Various supervisors of the conveyor and sorter system.

Example:
For the conveyor and sorter system, the combined plant is obtained through parallel composition of the two components and diagnoser. Next, this combined plant is refined with respect to the two requirements to obtain EFA G. Finally, a supervisor for the sorting system is synthesized in four different ways to demonstrate the differences between a supervisor obtained through the SSEFA(G) algorithm as described in [24], and the supervisor obtained through Algorithms 1 and 2. In the following figures, the equivalent FAs of these synthesized supervisors are shown, with the inner states representing nominal states in which $d^{jam} = F$ and the outer states representing post-failure states in which $d^{jam} = T$. In these figures noncritical states are green, critical states satisfying the failure-recoverability property are yellow, and critical states not satisfying this property are red.

First, a supervisor in which all critical states are allowed is synthesized using the SSEFA(G) algorithm from [24]. The resulting supervisor shown in Figure 13a includes a red state which is not failure-recoverable. This shows that allowing all critical states does not result in a machine safe supervisor. To obtain a machine safe supervisor, one could be synthesized using the SSEFA(G) algorithm, with as additional requirement that all critical states are forbidden. However, this results in the extremely

restrictive supervisor shown in Figure 13b. Using this supervisor, the controlled system can never start sorting, as that could lead to a critical state being reached.

Using the SSEFA(G) algorithm does not result in a supervisor which is both functional and which is failure-recoverable. Using Algorithm 1, the maximally permissive recoverable supervisor SSROB(G) for G is synthesized in which every state is failure-recoverable and fault-evitable. Critical states from which recovery to a noncritical state is possible are allowed (yellow), and states from which no recovery is possible are removed (red). The FA equivalent of supervisor SSROB(G) is shown in Figure 13c. The issue with supervisor SSROB(G) is that recovery to a noncritical state is not guaranteed. This is clearly shown in Figure 13c, where there could be an infinite loop of start sorting and stop sorting events in the post-failure states. This is solved by refining supervisor SSROB(G) using Algorithm 2, resulting in the fault-tolerant and failure-recovering supervisor SSRCS(G) as shown in Figure 13d. In this supervisor, all critical states are guaranteed to recover to a noncritical state, while nominal behaviour is unaltered.

6 Concluding Remarks

In this paper, a model-based design method is proposed for supervisors which are resilient to faults and failures. To accomplish this, two algorithm have been developed: one for the synthesis of a recoverable supervisor, and one for the synthesis of an optimally-recovering supervisor. Together, these algorithms can be used to synthesize the fault-tolerant and failure-recovering supervisor.

Algorithm 1 was developed to synthesize a maximally permissive recoverable supervisor. Like supervisors as described in [24], a supervisor ensures safety, controllability, and nonblocking. However, a recoverable supervisor also ensures two other properties. The first is failure-recoverability: if a failure occurs and the system ends up in a critical state, then the supervisor ensures that a quick recovery is always possible to a noncritical state. The second is that of fault-evitability: a fault may never be inevitable for the system to progress. A maximally permissive recoverable supervisor is a recoverable supervisor, such that plant behaviour is restricted as little as possible.

Algorithm 2 was developed to modify the maximally permissive recoverable supervisor obtained through Algorithm 1, such that a fault-tolerant and failure-recovering supervisor is obtained. Such a supervisor guarantees that the controlled system will always recover to a noncritical state as quickly as possible, and which disables controllable transitions to critical states. Just as the recoverable supervisor, the fault-tolerant and failure-recovering supervisor satisfies safety, controllability, nonblocking, failure-recoverability and fault-evitability. Although the supervisor which is synthesized with Algorithm 2 is not maximally permissive, the nominal behaviour imposed by the fault-tolerant and failure-recovering supervisor is restricted as little as possible.

Finally, a structured design method has been proposed, with a focus on modularity and clearness of the design. Modelling of the system is made intuitive, with the use of state-based expressions, the decomposition of plant model into diagnoser, component and physical relations, and the switching between nominal and post-fault system states with the use of variables. Following the design steps and using the developed synthesis algorithms, a supervisor for a discrete-event system can be designed which is resilient to faults and failures.

Currently, the biggest downside of the proposed design method is that it is not yet integrated with a tooling which allows for simulation, verification, real-time testing and realization of the supervisor. The next step in the development of the proposed design method should be integration of the fault-tolerant and failure-recovering supervisor synthesis procedure into such a tooling. A good candidate would be the *Compositional Interchange Format* (CIF) [27], as it supports both EFAs and the state-based expressions used in the proposed design method.

Another issue is the efficiency of the synthesis algorithms. As discussed in [28], the supervisory control problem is $\mathcal{NP}$-hard. Attempting to synthesize a fault-tolerant and failure-recovering supervisor for a complex system might be infeasible due to the state-space exploding. Solutions to this are discussed in [29] and [2]. One option is to create a supervisory controller with an architecture which is more efficient to synthesize, such as decentralized, modular, compositional or hierarchical supervisor synthesis. Another option is to use more efficient data structures for synthesis such as Binary Decision Diagrams (BDDs) [30]. The latter was achieved for the algorithm as described in [24], which is very similar to Algorithm 1.

Finally, after the above issues have been resolved, the next challenge would be to apply the proposed design method to an industrial case. Only then can it be fully evaluated if the proposed method succeeds in bridging the gap between theory and application.

REFERENCES

[1] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*, ser. Communications and Control Engineering. Springer International Publishing, 2019.

[2] J. Zaytoon and B. Riera, "Synthesis and implementation of logic controllers – A review," *Annual Reviews in Control*, vol. 43, pp. 152–168, 2017.

[3] L. Swartjes, D. A. van Beek, W. J. Fokkink, and J. A. W. M. van Eekelen, "Model-based design of supervisory controllers for baggage handling systems," *Simulation Modelling Practice and Theory*, vol. 78, pp. 28–50, 2017.

[4] M. A. Reniers and J. M. Van de Mortel-Fronczak, "An Engineering Perspective on Model-Based Design of Supervisors," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 257–264, 2018.

[5] S. T. J. Forschelen, J. M. van de Mortel-Fronczak, R. Su, and J. E. Rooda, "Application of supervisory control theory to theme park vehicles," *Discrete Event Dynamic Systems*, vol. 22, pp. 511–540, 2012.

[6] R. Fritz and P. Zhang, "Overview of fault-tolerant control methods for discrete event systems," *IFAC-PapersOnLine*, vol. 51, no. 24, pp. 88–95, 2018.

[7] J. Gertler, "Fault Detection and Diagnosis," *Encyclopedia of Systems and Control*, pp. 1–7, 2014.

[8] K. Cho and J. Lim, "Synthesis of fault-tolerant supervisor for automated manufacturing systems: a case study on photolithographic process," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 348–351, 1998.

[9] J. C. M. Baeten, J. M. Van de Mortel-Fronczak, and J. E. Rooda, "Integration of Supervisory Control Synthesis in Model-Based Systems Engineering," in *Complex Systems. Studies in Systems, Decision and Control*. Springer, Cham, 2016, vol. 55, pp. 39–58.

[10] R. R. H. Schiffelers, R. J. M. Theunissen, D. A. Van Beek, and J. E. Rooda, "Model-Based Engineering of Supervisory Controllers using CIF," *Electronic Communications of the EASST*, vol. 21, 2010.

[11] T. Korssen, V. Dolk, J. van de Mortel-fronczak, M. Reniers, and M. Heemels, "Systematic Model-Based Design and Implementation of Supervisors for Advanced Driver Assistance Systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 533–544, 2018.

[12] R. J. M. Theunissen, R. R. H. Schiffelers, D. A. Van Beek, and J. E. Rooda, "Supervisory control synthesis for a patient support system," in *2009 European Control Conference (ECC)*. IEEE, 2009, pp. 4647–4652.

[13] J. Zaytoon and S. Lafortune, "Overview of fault diagnosis methods for Discrete Event Systems," *Annual Reviews in Control*, vol. 37, pp. 308–320, 2013.

[14] F. F. H. Reijnen, M. A. Reniers, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Structured Synthesis of Fault-Tolerant Supervisory Controllers," *IFAC-PapersOnLine*, vol. 51, no. 24, pp. 894–901, 2018.

[15] T. Moor, "A discussion of fault-tolerant supervisory control in terms of formal languages," *Annual Reviews in Control*, vol. 41, pp. 159–169, 2016.

[16] J. Jiang and X. Yu, "Fault-tolerant control systems: A comparative study between active and passive approaches," *Annual Reviews in Control*, vol. 36, no. 1, pp. 60–72, 2012.

[17] T. Wittmann, J. Richter, and T. Moor, "Fault-Tolerant Control of Discrete Event Systems based on Fault-Accommodating Models," in *8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, 2012.

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

[19] Q. Wen, R. Kumar, J. Huang, and H. Liu, "A framework for fault-tolerant control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 53, no. 8, pp. 1839–1849, 2008.

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

[21] A. C. van Hulst, M. A. Reniers, and W. J. Fokkink, "Maximally Permissive Controlled System Synthesis for Modal Logic," in *SOFSEM 2015: Theory and Practice of Computer Science*, vol. 8939, 2015.

[22] R. Ziller and K. Schneider, "Combining Supervisor Synthesis and Model Checking," *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 2, pp. 331–362, 2005.

[23] R. Diekmann and D. Weidemann, "Event Enforcement in the Context of the Supervisory Control Theory," *Proceedings of the 18th International Conference on Methods and Models in Automation and Robotics*, pp. 783–788, 2013.

[24] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, "Nonblocking and Safe Control of Discrete-Event Systems Modeled as Extended Finite Automata," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, 2011.

[25] M. Sköldstam, K. Åkesson, and M. Fabian, "Modeling of Discrete Event Systems using Finite Automata With Variables," in *46th IEEE Conference on Decision and Control*, 2007, pp. 3387–3392.
[26] J. Zaytoon and V. Carré-Ménétrier, "Synthesis of control implementation for discrete manufacturing systems," *International Journal of Production Research*, vol. 39, pp. 329–345, 2001.
[27] D. A. van Beek, W. J. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, and M. A. Reniers, "CIF 3: Model-Based Engineering of Supervisory Controllers," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 8413 LNCS. Springer, Berlin, Heidelberg, 2014, pp. 575–580.
[28] P. Gohari and W. M. Wonham, "On the complexity of supervisory control design in the RW framework," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 30, no. 5, pp. 643–652, 2000.
[29] S. Miremadi and B. Lennartson, "Symbolic On-the-Fly Synthesis in Supervisory Control Theory," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 5, pp. 1705–1716, 2016.
[30] Z. Fei, S. Miremadi, K. Åkesson, and B. Lennartson, "Efficient symbolic supervisor synthesis for extended finite automata," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2368–2375, 2014.

APPENDIX A
PROOF OF CORRECTNESS OF ALGORITHM 1

In this appendix the correctness of Algorithm 1 is proven. This proof is based on Section IV.B from [24]. Given automaton G, SSROB(G) is the recoverable supervisor of G obtained by applying Algorithm 1 to G. The correctness of Algorithm 1 is proven by providing proofs for Proposition 1 and Theorems 4.1 and 4.2. Proposition 1 shows that the algorithm terminates in a finite number of steps. Theorem 4.1 shows that SSROB(G) is a nonblocking, safe, controllable, failure-recoverable and fault-evitable supervisor of G. Finally, Theorem 4.2 shows that SSROB(G) is the maximally permissive recoverable supervisor of G.

A.1 Proof of termination

Proposition 1 (Termination of Algorithm 1)**.** *Given EFA G with forbidden locations L_x and noncritical states Q_λ, the computation of SSROB(G) terminates in a finite number of steps.*

Proof. The proof of termination of execution is nearly identical to Proposition 1 in [24]. The number of reachable states in G is finite and is at most $|Q|$. It can be shown that each loop in Algorithm 1 terminates:

(a) Inside every iteration over j, the iteration over k loops as long as the robustness predicate of at least one state changes from F to T every iteration (lines 8-11). Remark 1 shows that the value of a robustness predicate of a state can only change once during the iterations over k. Conclusion: the iteration over k terminates in at most $|Q|$ steps.

(b) Inside every iteration over j, the iteration over i loops as long as the bad-state predicate of at least one state changes from F to T every iteration (lines 15-18). Remark 1 shows that the value of a bad-state predicate of a state can only change once during the iterations over i. Conclusion: the iteration over i terminates in at most $|Q|$ steps.

(c) The iteration over j loops as long as the guard of at least one transition changes from T to F every iteration (lines 19-21). Remark 2 and (line 19) show that a guard can only change due to a changing bad-state predicate. Remark 2 shows that the value of a bad-state predicate of a state can only change once during the iterations over j. Conclusion: the iteration over j terminates in at most $|Q|$ steps.

The above steps prove that Algorithm 1 terminates in a finite number of steps. Iteration j terminates in $O(|Q|)$, and the iterations i and k inside j terminate in $O(|Q|)$. The complexity of Algorithm 1 is $O(|Q|^2)$. □

A.2 Proof of nonblocking, safety, controllability, failure-recoverability and fault-evitability

The proof of nonblocking, safety, controllability, failure-recoverability and fault-evitability of SSROB(G) is given in Theorem 4.1. Proposition 1 shows that Algorithm 1 terminates in a finite number of iterations, with N defined as the terminal value of j.

Lemma A.1. *Given automaton G with forbidden locations L_x and noncritical states Q_λ. Suppose that $j = N$ when Algorithm 1 terminates. Every state $q \in Q^G$ for which the bad-state predicate $B^N(q)$ equals T is either unreachable in SSROB(G), or an initial bad-state predicate $B^N(q_0) = T$ exists in G:*

$$\Big(\forall q \in Q^G\Big)\Big[B^N(q) = T \implies q \notin Q^{SSROB(G)}\Big] \vee \Big(\exists q_0 \in Q_0\Big)\Big[B^N(q_0) = T\Big]$$

Proof. This proof is based on the proof for Lemma 1 in [24]. The following deduction provides a proof for a path to a state $q \in Q^G$ to show the state is made unreachable in SSROB(G). Generalization can be accomplished by applying the following procedure to every possible path $p^G_{q_0 \to q}$ from every possible state in automaton G, with $q_0 \in Q_0$ and $q \in Q^G$.

Assume there is a state $q \in Q^G$ for which $B^N(q) = T$. As shown in Figure 14, state q is reached through a path $\eta = p^G_{q_0 \to q}$. Path η is characterized as a path with (from end to beginning): $0..n$ uncontrollable transitions to reach q from q', $0..1$ controllable transitions θ_c to reach q' from q'' (the last controllable transition in η) and $0..m$ transitions (which can be both controllable and uncontrollable) to reach q'' from q_0.

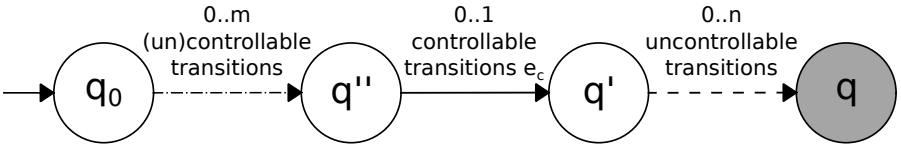


Figure 14: An illustration of path η [24].

Suppose the bad-state predicate $B^{j,i}(q)$ changes value from F to T at $j = U, i = V$ with $U < N$ (no bad-state predicate changes on $j \geq N$, or the algorithm would not terminate at $j = N$). Remark 1 shows that $B^{j,i}(q) = F$ for any iterations before $j = U, i = V$, and $B^{j,i}(q) = T$ for any iterations after $j = U, i = V$. Figure 14 shows three different scenarios for path η: at least one controllable and no uncontrollable transitions, at least one controllable and at least one uncontrollable transition, and lastly, no controllable transition. Below is proof that in the first two scenarios q is unreachable in SSROB(G) when $B^N(q) = T$, while for the third scenario it is proven that there is an initial state $q_0 \in Q_0$ which is a bad-state, so $B^N(q_0) = T$.

Case 1: for at least one controllable transition and no uncontrollable transitions in path η

If there is at least one controllable transition in path η, and no uncontrollable transitions ($n = 0$), then $q' = q$. As mentioned before, the bad-state predicate for q changes to T on iteration $j = U, i = V$, and stays T for further iterations. This leads to $B^U(q) = T$ on (line 18)
$B^U(q) = T$, which leads to the following at (line 19) of the algorithm:

$$g_{e_c}^{U+1}(d_{q''}) \quad = \quad g_{e_c}^{U}(d_{q''}) \wedge \neg B^U(q) \quad = \quad g_{e_c}^{U}(d_{q''}) \wedge F \quad = \quad F$$

Due to Remark 2, guard $g_{e_c}^{j}(d_{q''})$ stays F for iterations $j > U$, so $g_{e_c}(d_{q''}) = F$. Thus q is proven to be unreachable in SSROB(G).

Case 2: for at least one controllable transition and at least one uncontrollable transition in path η

If there is at least one controllable transition in path η, and at least one uncontrollable transition between q' and q'' ($n > 0$), then:

- First define path μ as part of path η:

$$\mu = p^{G}_{q' \xrightarrow{\Sigma_u} q} = q_1 \xrightarrow{\sigma_{u_1}} \cdots \xrightarrow{\sigma_{u_{r-1}}} q_r \xrightarrow{\sigma_{u_r}} \cdots \xrightarrow{\sigma_{u_n}} q_{n+1} \qquad \text{with } q_1 = q' \text{ and } q_{n+1} = q.$$

- The transition relation between q_r and q_{r+1} is: $e_r = (l_r, \sigma_{u_r}, g_r, f_r, l_{r+1})$. Because σ_{u_r} is uncontrollable, the guard of any transition in μ stays true for all iterations of j: $g_r^j(d_{q_r}) = T$.
- On iteration $j = U, i = V$, the bad state predicate of state q changes, so: $B^{U,V}(q_{n+1}) = T$.
- Every state in μ has an uncontrollable transition to the next state in the path. Consequently, if any state in μ is flagged as a bad state, then the next iteration of i, the preceding state will also be flagged as a bad state. Thus, on iterations $j = U, i = V'$ (with $V' = V + 1, \ldots, V + n$, for which n is the number of uncontrollable transitions between q' and q), the bad-state predicate $B^{U,V'-1}(q_{r+1}) = T$ (with $r = n - V' - V + 1$). So according to (line 15):

$$B^{U,V'}(q_r) = \quad B^{U,V'-1}(q_r) \vee \left[g_r^U(d_{q_r}) \wedge B^{U,V'-1}(q_{r+1}) \right] \quad = \quad B^{U,V'-1}(q_r) \vee [T \wedge T] \quad = \quad T$$

- So no later than iteration $j = U, i = V+n$, the bad-state predicate of state q' changes: $B^{U,V+n}(q') = B^{U,V+n}(q_1) = T$. Consequently, due to Remark 1 and (line 18): $B^U(q') = T$.
- The guard of controllable transition e_c between q'' and q' changes on (line 19): $g_{e_c}^{U+1}(d_{q''}) = g_{e_c}^{U}(d_{q''}) \wedge \neg B^U(q') = g_{e_c}^{U}(d_{q''}) \wedge \neg T = F$. Consequently, due to Remark 2: $g_{e_c}^{N}(d_{q''}) = F$.

The result of $g_{e_c}^{N}(d_{q''}) = F$ is that every state in path μ, including state q, is unreachable in SSROB(G).

Case 3: for no controllable transition in path η

If there is no controllable transition in path η, but there is at least one uncontrollable transition ($n > 0$), then $q_0 = q'$. The same steps as for Case 2 are repeated, except for the last step as there are no controllable transition guards which can be changed. The algorithm will update the bad-state predicate to T for all states in path η, including the initial state q_0: $B^N(q_0) = B^{U,V+n}(q') = T$. If there are neither controllable nor uncontrollable transitions in path η, then $q_0 = q$ and the initial state is a bad-state by definition: $B^N(q) = B^N(q_0) = T$.

So to recap, q is made unreachable in SSROB(G) when $B^N(q) = T$ in the first two scenarios, and in the third scenario it is proven that there is an initial state $q_0 \in Q_0$ which is a bad-state $B^N(q_0) = T$.

□

Instead of confirming nonblockingness, fault-evitability and failure-recoverability of a supervisor G^s separately, Lemma A.2 specifies a condition which only holds if the supervisor satisfies all three. This condition will be referred to as robustness of the supervisor.

Lemma A.2 (Robustness)**.** *A supervisor G^s satisfies robustness (nonblocking, failure-recoverability and fault-evitability), if and only if there is a path from every reachable state $q \in Q^{G^s}$ to a marked, noncritical state $q_{m,\lambda} \in (Q_m \cap Q_\lambda)$, such that for the path:*

(I) No transition uses a fault event.
(II) All transitions originating from a critical state are controllable.
(III) All transitions originating from a noncritical state, terminate in a noncritical state.

This condition for a reachable state $q \in Q^{G^s}$ can be formalized as following:

$$\exists p^{G^s}_{q \to q_{m,\lambda}} \text{ s.t. } \left(q_{m,\lambda} \in Q_m \cap Q_\lambda\right) \wedge \left(\forall i : 1 \leq i < |p^{G^s}_{q \to q_{m,\lambda}}|, (\sigma_{e_i} \notin \Sigma_f) \wedge (q_i \in Q_\lambda \implies q_{i+1} \in Q_\lambda) \wedge (q_i \notin Q_\lambda \implies \sigma_{e_i} \in \Sigma_c)\right)$$

Proof. In Lemma A.2 it is claimed that a supervisor G^s satisfies nonblockingness, fault-evitability and failure-recoverability, if there exists a path in every reachable state $q \in Q^{G^s}$ such that (I), (II) and (III) hold. As seen in the condition in Lemma A.2, a state in this path being critical implies that the transition to the next state in the path is controllable, and a state in this path being noncritical implies that all subsequent states are also noncritical states. Let γ denote such a path, with first r transitions with controllable events till a noncritical state is reached, from which onwards there are n transitions with non-fault events between noncritical states, till a marked, noncritical state is reached. Figure 15 shows that path γ indeed satisfies all three properties. Also, it is shown in Figure 15 that path γ can be split in consecutive subpaths γ_1 and γ_2, with γ_1 the path from q to q_{λ_1} and γ_2 the path from q_{λ_1} to $q_{m,\lambda}$. Path γ is formally defined as:

$$\begin{aligned}\gamma = q \xrightarrow{\sigma_{c_1}} \cdots \xrightarrow{\sigma_{c_{t-1}}} q_t \xrightarrow{\sigma_{c_t}} \cdots \xrightarrow{\sigma_{c_r}} q_{\lambda_1} \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{p-1}} q_{\lambda_p} \xrightarrow{\sigma_p} \cdots \xrightarrow{\sigma_n} q_{m,\lambda},\\ \text{with } \sigma_{c_t} \in \Sigma_c \text{ and } q_t \notin Q_\lambda \text{ (for } t = 1..r),\\ \sigma_p \in (\Sigma - \Sigma_f) \text{ and } q_{\lambda_p} \in Q_\lambda \text{ (for } p = 1..n),\\ q_{m,\lambda} \in (Q_\lambda \cap Q_m).\end{aligned}$$

With the following definitions for states in path γ:

$$\begin{array}{llll} q = q_1, & q_{\lambda_1} = q_{r+1}, & q_{m,\lambda} = q_{\lambda_{n+1}} & \text{if } r > 0, n > 0\\ q = q_1, & q_{\lambda_1} = q_{m,\lambda} = q_{r+1}, & & \text{if } r > 0, n = 0\\ q = q_{\lambda_1}, & & q_{m,\lambda} = q_{\lambda_{n+1}} & \text{if } r = 0, n > 0\\ q = q_{\lambda_1} = q_{m,\lambda}, & & & \text{if } r = 0, n = 0 \end{array}$$

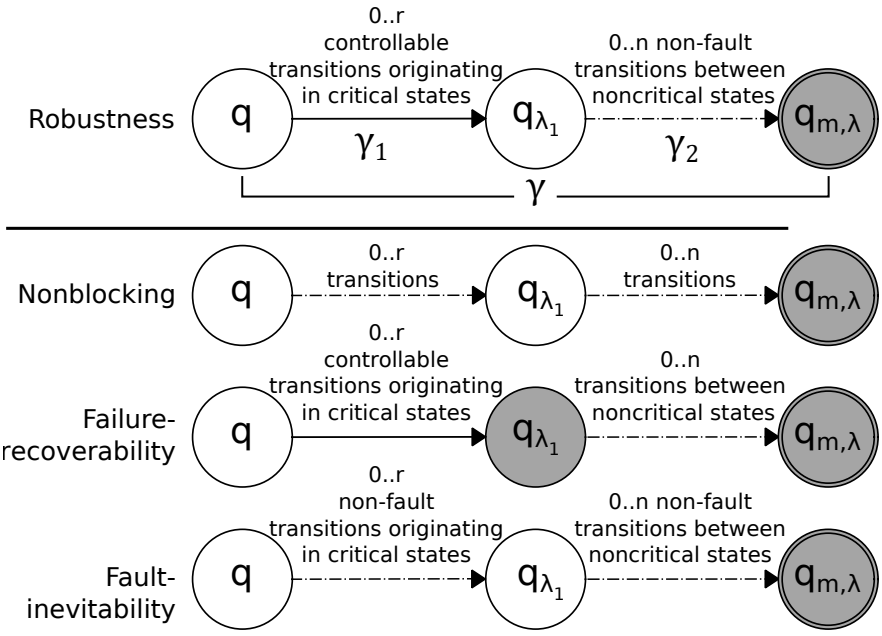


Figure 15: The path γ for robustness, and why nonblocking, failure-recoverability, and failure-evitability are satisfied in all states in γ.

The following steps prove how the existence of a path γ in every state $q \in Q^{G^s}$ indicates nonblocking, failure-recoverability and fault-evitability:

(a) For q to satisfy failure-recovery there needs to be a failure-recovery path from q to a noncritical state $q_{\lambda_1} \in Q_\lambda$. The failure-recovery path coincides with subpath γ_1, as seen in Figure 15. All states in this path are failure-recoverable.
(b) If q_{λ_1} satisfies fault-evitability, then q and every other state in γ_1 satisfy fault-evitability, as every transition in γ_1 is a non-fault transition originating from a critical state.
(c) If q_{λ_1} satisfies nonblocking, then q and every other state in γ_1 satisfies nonblocking as there is an enabled path to q.
(d) The first noncritical state after failure-recovery q_{λ_1} is a noncritical state. For it to satisfy fault-evitability, there must be a path to a noncritical marked state $q_{m,\lambda} \in (Q_\lambda \cap Q_m)$, with the path consisting of transitions between noncritical states, and using only non-fault events. This path coincides with subpath γ_2, as seen in Figure 15. Thus following (b), all states in γ satisfy fault-evitability.
(e) All states in γ_2 are noncritical, and are failure-recoverable by definition. Thus following (a), all states in γ are failure-recoverable.
(f) All states in γ_2 have a path towards a marked state, and thus are nonblocking by definition. Thus following (c), all states in γ are nonblocking.

Thus, if there exists a path γ in every state $q \in Q^{G^s}$, then G^s satisfies nonblocking, failure-recoverability and fault-evitability.

□

Next, with the use of Lemma A.2, Theorem 4.1 is proven.

Theorem 4.1 (Nonblockingness, safety, controllability, failure-recoverability and fault-evitability of SSROB(*G*))**.** *Given EFA G with forbidden locations L_x and noncritical states Q_λ, SSROB(G) is nonblocking, safe, controllable, failure-recoverable and fault-evitable as long as none of its initial states is a bad-state.*

Proof. The proof of the nonblockingness, safety, controllability, failure-recoverability and fault-evitability of SSROB(*G*) is based on the proof of Theorem 2 in [24].

Proof of safety:
Per Definition 4, a state $q \in Q^{\text{SSROB}(G)}$ is a safe state when $q \notin Q_x$. Lemma 1 shows that either all bad-states of G are made unreachable, or there is an initial bad-state in G:

$$\Big(\forall q \in Q^G\Big)\Big[B^N(q) = T \implies q \notin Q^{\text{SSROB}(G)}\Big] \vee \Big(\exists q_0 \in Q_0\Big)\Big[B^N(q_0) = T\Big]$$

In Algorithm 1, (line 12) shows that all unsafe states start out as bad states: $(\forall q_x \in Q_x)[B^{j,0}(q_x) = T]$. Remark 1 shows that the bad-state predicate of these states will stay T till terminal iteration $j = N$. This proves that either SSROB(*G*) is safe, or there is an initial bad-state.

Proof of controllability:
SSROB(*G*) is proven to be controllable if every state $q \in Q^{\text{SSROB}(G)}$ is controllable. Per Definition 4, controllability requires every q to be safe, which has been proven. Given that Q^G_{q,Σ_u} is the set of all states reachable with an uncontrollable event from state q. Then for q to be controllable it is also required that: $Q^G_{q,\Sigma_u} \subseteq Q^{\text{SSROB}(G)}$.

In Algorithm 1, guards of uncontrollable transitions are never modified (line 19), so all uncontrollable transitions enabled in $q \in Q^G$, are also enabled in $q \in Q^{\text{SSROB}(G)}$. This means that all states in Q^G_{q,Σ_u} are reachable in SSROB(*G*), which proves $Q^G_{q,\Sigma_u} \subseteq Q^{\text{SSROB}(G)}$ and consequently that SSROB(*G*) is controllable. So to conclude, either automaton G contains an initial bad-state, or all states in SSROB(*G*) are safe and controllable.

Proof of nonblockingness, failure-recoverability and fault-evitability:

Per Definition 4 and Lemma A.2, if a supervisor is nonblocking, failure-recoverable and fault-evitable, then robustness must be satisfied in every reachable state of G^s. A state $q \in Q^{\text{SSROB}(G)}$ satisfies robustness when there exists a path $p^{\text{SSROB}(G)}_{q \to q_m}$ to a noncritical, marked state $q_{m,\lambda} \in Q_m \cap Q_\lambda$, using only non-fault events and while never transitioning from a noncritical to a critical state, with transitions originating from a critical state always using controllable events.

Let $\mathcal{P}^{\text{SSROB}(G)}(q)$ denote all existing paths from state q for which no transition uses a fault event, for which there never is a transition from a noncritical state to a critical state, and for which a transition

from a critical state always uses a controllable event. For a path $p^{\text{SSROB}(G)}_{q\to q''}$, let q_i and e_i denote the ith state and transition in the path respectively.

$$\mathcal{P}^{\text{SSROB}(G)}(q) = \Big\{p^{\text{SSROB}(G)}_{q\to q''} \big| (q'' \in Q^{\text{SSROB}(G)}) \wedge \Big(\forall i : 1 \leq i < |p^{\text{SSROB}(G)}_{q\to q''}|, \\ \big((\sigma_{e_i} \notin \Sigma_f) \wedge (q_i \in Q_\lambda \implies q_{i+1} \in Q_\lambda) \wedge (q_i \notin Q_\lambda \implies \sigma_{e_i} \in \Sigma_c)\big)\Big)\Big\}$$

If the algorithm is correct, a state which does not satisfy robustness should always be made unreachable. Suppose a state $q \in Q^{\text{SSROB}(G)}$ does not satisfy robustness, but is still reachable when the algorithm terminates at $j = N$. Let $Q_{\mathcal{P}_q}$ denote all states in paths $\mathcal{P}^{\text{SSROB}(G)}(q)$. If q does not satisfy robustness, then $Q_{\mathcal{P}_q} \cap Q_m \cap Q_\lambda = \emptyset$. Lemma A.1 implies that if q is reachable at $j = N$, then either $B^N(q) = F$, or an initial bad state exists. Proposition 1 implies that if $B^N(q) = F$ (so when the algorithm terminates), then $B^{N+1}(q) = B^N(q) = F$. Presume Algorithm 1 is ran for an additional iteration $j = N + 1$. If there are no initial bad-states, then none of the bad-state predicates can change from iteration $j = N$ to $J = N + 1$, or else Lemma A.1 or Proposition 1 are violated.

(a) Equation $Q_{\mathcal{P}_q} \cap Q_m \cap Q_\lambda = \emptyset$ and (line 5) (for $j = N + 1, k = 0$) imply:

$$\forall q' \in Q_{\mathcal{P}_q}, \quad R^{N+1,0}(q') = F$$

(b) For $\forall q' \in Q_{\mathcal{P}_q}$, only transitions $e \in E_{q'}$ are evaluated on (line 8), and all terminal states of those transitions are states in $Q_{\mathcal{P}_q}$:

$$(\forall q' \in Q_{\mathcal{P}_q})(e \in E_{q'}), \qquad \bar{q}_{e,d_{q'}} \in Q_{\mathcal{P}_q},$$

with $E_{q'} = \Big\{\forall e \in E | (o_e = l_{q'}) \wedge (\sigma_e \notin \Sigma_f) \wedge (q' \in Q_\lambda \implies \bar{q}_{e,d_{q'}} \in Q_\lambda) \wedge (q' \notin Q_\lambda \implies \sigma_e \in \Sigma_c)\Big\}$.

(c) According to (a) for $k = 0$: $\bar{q}_{e,d_{q'}} \in Q_{\mathcal{P}_q}, R^{N+1,0}(q') = F$. So for every subsequent iteration $k > 0$ the predicates of all reachable terminal states are F, which leads to:

$$\forall q' \in Q_{\mathcal{P}_q}, \quad R^{N+1,k}(q') = \quad F \ \vee \bigvee_{\forall e \in E_{q'}} \Big[g_e^{N+1}(d_{q'}) \wedge R^{N+1,k-1}(\bar{q}_{e,d_{q'}})\Big] \quad = \\ F \ \vee \bigvee_{\forall e \in E_{q'}} \Big[g_e^{N+1}(d_{q'}) \wedge F\Big] \quad = \quad F$$

(d) When the predicate R of all states in $Q_{\mathcal{P}_q}$ remains unchanged for iterations ($j = N + 1, k > 0$), then according to (line 11):

$$\forall q' \in Q_{\mathcal{P}_q}, \quad R^{N+1}(q') = \quad F$$

(e) This results in the following initial bad-state predicate according to (line 12):

$$\forall q' \in Q_{\mathcal{P}_q}, \quad B^{N+1,0}(q') = \quad \neg R^{N+1}(q') \vee B^N(q') \quad = \quad \neg F \vee B^N(q') \quad = \quad T$$

(f) Following Remark 1, every subsequent iteration ($j = N + 1, i > 0$):
$\forall q' \in Q_{\mathcal{P}_q}, \ B^{N+1,i}(q') = T$, so:

$$\forall q' \in Q_{\mathcal{P}_q}, \quad B^{N+1}(q') = \quad B^{N+1,i}(q') \quad = \quad T$$

The above would conflict with either Lemma A.1 or Proposition 1. According to Lemma A.1, state q should be made unreachable, or an initial bad-state should exist, which means that q is reachable on iteration $j = N$, but unreachable on iteration $j = N + 1$. However, this contradicts that the algorithm would terminate on iteration $j = N$. This proves that if there is no initial bad-state in SSROB(G), then state q cannot be reachable in SSROB(G) if it is does not satisfy nonblockingness, failure-recoverability and fault-evitability.

□

A.3 Proof of maximal permissiveness

In this subsection the proof of maximal permissiveness of SSROB(G) is given.

Theorem 4.2 (Maximal permissiveness of SSROB(G))**.** *Given EFA G with forbidden states Q_x and noncritical states Q_λ, SSROB(G) is the maximally permissive recoverable supervisor of G.*

Proof. The proof of this theorem is based on the proof of Theorem 3 in [24]. Let G' be a recoverable supervisor of G, with none of the initial states in G being bad-states. Following Definition 5, SSROB(G) is the maximally permissive supervisor of G when all possible supervisors of G are a subautomaton of SSROB(G): $\forall G' \in \mathcal{A}(G), \ G' \preceq$ SSROB(G). So it needs to be proven that a supervisor G' is a subautomaton of SSROB(G): $G' \preceq$ SSROB(G), which is true if $Q^{G'} \subseteq Q^{\text{SSROB}(G)}$. To prove this it needs

to be shown that a state $q \in Q^{G'}$ is not made unreachable in SSROB(G), so: $q \in Q^{G'} \Rightarrow q \in Q^{\text{SSROB}(G)}$.

Recall that robustness is satisfied if nonblocking, failure-recoverability and fault-evitability are satisfied. Robustness is satisfied if in every state of the supervisor there exists a path as described in the proof of Lemma A.2 and depicted in Figure 15. Assume without loss of generality that γ is the shortest path from q to a state $q_{m,\lambda} \in Q_m \cap Q_\lambda$, such that robustness is satisfied (if there is a path from q which satisfies robustness, then one of them must be the shortest). All states in γ are reachable, so all guards in γ are enabled. Recall that path γ is formally defined as follows:

$$\begin{aligned}\gamma = q \xrightarrow{\sigma_{c_1}} \cdots \xrightarrow{\sigma_{c_{t-1}}} q_t \xrightarrow{\sigma_{c_t}} \cdots \xrightarrow{\sigma_{c_r}} q_{\lambda_1} \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{p-1}} q_{\lambda_p} \xrightarrow{\sigma_p} \cdots \xrightarrow{\sigma_n} q_{m,\lambda},\\ \text{with } \sigma_{c_t} \in \Sigma_c \text{ and } q_t \notin Q_\lambda \text{ (for } t = 1..r),\\ \sigma_p \in (\Sigma - \Sigma_f) \text{ and } q_{\lambda_p} \in Q_\lambda \text{ (for } p = 1..n),\\ q_{m,\lambda} \in (Q_\lambda \cap Q_m).\end{aligned}$$

With the following definitions for states in path γ:

$$\begin{array}{llll} q = q_1, & q_{\lambda_1} = q_{r+1}, & q_{m,\lambda} = q_{\lambda_{n+1}} & \text{if } r > 0, n > 0\\ q = q_1, & q_{\lambda_1} = q_{m,\lambda} = q_{r+1}, & & \text{if } r > 0, n = 0\\ q = q_{\lambda_1}, & & q_{m,\lambda} = q_{\lambda_{n+1}} & \text{if } r = 0, n > 0\\ q = q_{\lambda_1} = q_{m,\lambda}, & & & \text{if } r = 0, n = 0 \end{array}$$

The computation of SSROB(G) for supervisor G' is as following and shows that q indeed is not made unreachable:

(a) For the initial iteration, only predicate R of state $q_{m,\lambda}$ is T. Predicates for all other states in γ are F. Remark 1 shows that any predicate R which changes to T, will stay T for subsequent iterations of k. So for $j = 0, k = 0$:

$$\begin{aligned} R^{0,0}(q_m) = \quad & T\\ \forall q_{\lambda_p} \text{ in } \gamma \ (\text{for } p = 1, ..., n), \quad R^{0,0}(q_{\lambda_p}) = \quad & F \quad (\text{if } q_{\lambda_1} \neq q_{m,\lambda})\\ \forall q_t \text{ in } \gamma \ (\text{for } t = 1, ..., r), \quad R^{0,0}(q_t) = \quad & F \quad (\text{if } q_1 \neq q_{m,\lambda}) \end{aligned}$$

(b) For iterations $j = 0$, k (for $k = 1, ..., n$) predicate R of state q_{λ_p} (with $p = n - k + 1$) changes to T because: $R^{0,k}(q_{\lambda_{p+1}}) = T$, $q_{\lambda_p} \in Q_\lambda$, $q_{\lambda_{p+1}} \in Q_\lambda$, $\sigma_p \notin \Sigma_f$ and $g_p^0(d_{q_{\lambda_p}}) = T$, so:

$$R^{0,k}(q_{\lambda_p}) = \quad R^{0,k-1}(q_{\lambda_p}) \vee [g_p^0(d_{q_{\lambda_p}}) \wedge R^{0,k-1}(q_{\lambda_{p+1}})] \quad = \quad F \vee [T \wedge T] \quad = \quad T$$

(c) So on iteration $j = 0, k = n$, predicate R of state $q_{\lambda_1} = q_{r+1}$ has changed to T. Next, for iterations $j = 0$, k (for $k = n + 1, ..., n + r$) predicate R of state q_t (with $t = r + n - k + 1$) changes to T because: $R^{0,k}(q_{t+1}) = T$, $q_t \notin Q_\lambda$, $\sigma_t \in \Sigma_c$ and $g_t^0(d_{q_t}) = T$, so:

$$R^{0,k}(q_t) = \quad R^{0,k-1}(q_t) \vee [g_t^0(d_{q_t}) \wedge R^{0,k-1}(q_{t+1})] \quad = \quad F \vee [T \wedge T] \quad = \quad T$$

(d) So on iteration $j = 0, k = n + r$, for state q, predicate $R^{0,k}(q)$ has changed to T, which it stays for further iterations of k. Which leads to $R^0(q) = T$ at (line 11) of the algorithm. Just as state q, every state in G' is defined as robust, so per definition:

$$\forall q' \in Q^{G'}, \quad R^0(q') = \quad T$$

(e) Supervisor G' is safe, thus $\forall q' \in Q^{G'}, \ q' \notin Q_x$. So for $j = 0, i = 0$ on (line 12), $B^{0,0}(q) = F$. In general for all states of G':

$$\forall q' \in Q^{G'}, \quad B^{0,0}(q') = \quad F$$

(f) Next, supervisor G' is controllable. Suppose there is a transition $e_u \in E$ in G from state q'' to $\bar{q}_{e_u,q''}$, with an uncontrollable event $\sigma_{e_u} \in \Sigma_u$, so: $q'' \xrightarrow{\sigma_{e_u}} \bar{q}_{e_u,q''}$. Then the following implication is true in controllable supervisor G':

$$q'' \in Q^{G'} \Rightarrow \bar{q}_{e_u,q''} \in Q^{G'}$$

(g) The results of (e) and (f) imply that in (line 15) no states are reachable with a bad-state predicate which is T, so for further iterations of i, all bad state predicates will remain F. So for $j = 0, i > 0$:

$$\forall q' \in Q^{G'}, \quad B^{0,i}(q') = \quad F \vee \bigvee_{\{e \in E | o_e = l_{q'}, \sigma_e \in \Sigma_u\}} [g_e^j(d_{q'}) \wedge F] \quad = \quad F$$

So following (line 18) for $j = 0$:

$$\forall q' \in Q^{G'}, \quad B^0(q') = \quad F$$

(h) Next, (line 19) and (g) result in all guards being T for $j = 0$, so:

$$(\forall e \in E)(\forall q' \in Q^{G'} : o_e = l_{q'}), \qquad g_e^1(d_{q'}) = \quad T \wedge \neg F \quad = \quad T$$

None of the guards in G' change, so iterations $j > 0$ give the same results in (a-f). Therefore, all guards in G' are still enabled when the algorithm terminates at $j = N$:

$$(\forall e \in E)(\forall q' \in Q^{G'} : o_e = l_{q'}), \qquad g_e^N(d_{q'}) = \quad T$$

(i) And finally, the algorithm outputs the following and then terminates:

$$\text{SSROB}(G) = (L, D, \Sigma, E^{\text{SSROB}(G)}, L_0, D_0, L_m),$$
$$\text{with } \forall e \in E : (o_e, \sigma_e, g_e^N, f_e, t_e) \in E^{\text{SSROB}(G)}.$$

To conclude, state $q \in G'$ is not made unreachable in SSROB(G), so: $Q^{G'} \subseteq Q^{\text{SSROB}(G)}$, and subsequently $\forall G' \in \mathcal{A}(G),\ G' \preceq \text{SSROB}(G)$. This proves that SSROB(G) is the maximally permissive recoverable supervisor of G. □

APPENDIX B
PROOF OF CORRECTNESS OF ALGORITHM 2

In this appendix the correctness of Algorithm 2 is proven. First Theorem 4.3 is proven, which states that the optimally-recovering supervisor derived from a recoverable supervisor, is safe, controllable, nonblocking, fault-evitable and failure-recoverable. This proof implies that using supervisor SSROB(G) as input for Algorithm 2 results in SSRCS(G) being a fault-tolerant and failure-recovering supervisor. Next, termination of Algorithm 2 is proven in Proposition 2. Finally, it is proven that SSRCS(G) is a correct optimally-recovering supervisor of G^s.

B.1 Proof of the optimally-recovering supervisor being safe, controllable, nonblocking, fault-evitable and failure-recoverable

Theorem 4.3 (Safety, controllability, nonblockingness, fault-evitability and failure-recoverability of G^{rcs})**.** *Given recoverable supervisor G^s, and optimally-recovering supervisor G^{rcs}. If G^s is safe, nonblocking, controllable, failure-recoverable and fault-evitable, then G^{rcs} is also safe, nonblocking, controllable, failure-recoverable and fault-evitable.*

Proof. Below is the proof of Theorem 4.3. Recoverable supervisor G^s satisfies safety, controllability, nonblockingness, fault-evitability and failure-recoverability by definition. It needs to be proven that all states in G^{rcs} also satisfy these properties. As defined in Definition 7, G^{rcs} is obtained by making guards on transitions stricter. A transitions in G^s is restricted if it: does not terminate in a noncritical state, has a non-decreasing failure-recovery distance, and is controllable.

- Safety is satisfied in G^{rcs}, as all states in G^s are defined to be safe states, and making guards stricter cannot lead to a forbidden state becoming reachable.
- Controllability is satisfied in G^{rcs}. Recoverable supervisor G^s is controllable, and no guards on uncontrollable transitions are modified, nor do any states become unsafe in G^{rcs}.
- Failure-recoverability is satisfied in G^{rcs}. For a state $q \in G^s$, transitions in the shortest failure-recovery path(s) remain unrestricted in G^{rcs}, as for every transition in such path the failure-recovery distance is always decreasing: $\omega^{G^s}(q_t) > \omega^{G^s}(q_{t+1})$.
- Fault-evitability is satisfied in G^{rcs}. Lets say the shortest failure-recovery path which stays enabled in G^{rcs}, as discussed above, terminates in noncritical state $q_{\lambda_1} \in Q_\lambda$. State q_{λ_1} satisfies fault-evitability in G^s and is a noncritical state. This means there is a path in G^s from q_{λ_1} to a marked, noncritical state $q_{m,\lambda} \in (Q_\lambda \cap Q_m)$, with only non-fault transitions between noncritical states in this path. If a transition to a noncritical state is enabled in G^s, then it stays enabled in G^{rcs}. Henceforth, q_{λ_1} satisfies fault-evitability in G^{rcs}. All transitions in a failure-recoverable path are controllable and do not originate in a noncritical state, thus if q_{λ_1} is fault-evitable in G^{rcs}, then q is also fault-evitable in G^{rcs}.
- Nonblockingness is satisfied in G^{rcs}. Fault-evitability being satisfied proves that there is an enabled path from state q to a marked state.

So if G^s is safe, controllable, nonblocking, fault-evitable and failure-recoverable, then G^{rcs} is also safe, controllable, nonblocking, fault-evitable and failure-recoverable. □

B.2 Proof of termination

Proposition 2 (Termination of Algorithm 2)**.** *Given EFA G^s with noncritical states Q_λ, the computation of SSRCS(G) terminates in a finite number of steps.*

Proof. The proof of termination of Algorithm 2 is as follows. Initially $V^0 = Q \setminus Q_\lambda$, thus the number of reachable states is finite and is at most $|Q|$. The iteration over i continues as long as $W^i \neq \emptyset$, and for W^i to be nonempty there must be at least one state in the set. On (line 7) only states which are in V^{i-1} can be added to W^i. For the next iteration $V^i = V^{i-1} \setminus W^i$, which means that V^i decreases by at least one state every iteration of i. Conclusion: the iteration over i terminates in at most $|Q|$ steps. After the iteration over i the guards are updated, and the algorithm terminates. The complexity of Algorithm 2 is $O(|Q|^2)$.

□

B.3 Proof of SSRCS(*G*) being the optimally-recovering supervisor of G^s

Theorem 4.4 (SSRCS(*G*) is the optimally-recovering supervisor of SSROB(*G*))**.** *Given EFA G^s with noncritical states Q_λ, SSRCS(G) is the optimally-recovering supervisor of G^s.*

Proof. To prove Theorem 4.4, it needs to be proven that the correct transitions are restricted in SSRCS(*G*) per Definition 7, and that all other transitions are unaltered in SSRCS(*G*).

- First, for a state $q \in Q$ $\omega^{G^s}(q)$ is the defined failure-recovery distance, and $\hat{\omega}(q)$ is the failure-recovery distance as calculated in the algorithm. It needs to be proven that $\forall q \in Q,\ \omega^{G^s}(q) = \hat{\omega}(q)$. Suppose for all finite k, the set of all states $q \in Q$ for which $\omega^{G^s}(q) = k$ is denoted by W^k. In the algorithm, the set of all states $q \in Q$ for which $\hat{\omega}(q) = k$ is denoted by $\hat{W}^k$.

- First, these sets must be calculated for $k = 0$:

  $W^0 = \{q \in Q | \omega^{G^s}(q) = 0\} = Q_\lambda$ following the above definition

  $\hat{W}^0 = Q_\lambda$ following (line 4) in Algorithm 2

  Which proves that $W^0 = \hat{W}^0$.

- Next, these set must be calculated for all $k \in \mathbb{N}_1$. This can be done with proof by induction:
  - Following (lines 4-10), the set of all states for which $\hat{\omega}(q) > k$ is: $\hat{V}^k = \hat{V}^{k-1} \backslash \hat{W}^k = Q \backslash \bigcup_{j=0}^{k} \hat{W}^j$. Then for a value of k:

    $W^k = \{q \in Q | \omega^{G^s}(q) = k\}$

    $\hat{W}^k = \left\{ q \in (Q \setminus \bigcup_{j=0}^{k-1} \hat{W}^j) \;\middle|\; \exists e \in E \text{ s.t. } \left((o_e = l_q) \wedge (\sigma_e \in \Sigma_c) \wedge (g_e(d_q) = T) \wedge (\bar{q}_{e,d_q} \in \hat{W}^{k-1})\right) \right\}$
  - Suppose $W^k = \hat{W}^k$ is correct, then it needs to be proven that $W^{k+1} = \hat{W}^{k+1}$:

    $W^{k+1} = \{q \in Q | \omega^{G^s}(q) = k+1\}$

    $\hat{W}^{k+1} = \left\{ q \in (Q \setminus \bigcup_{j=0}^{k} \hat{W}^j) \;\middle|\; \exists e \in E \text{ s.t. } \left((o_e = l_q) \wedge (\sigma_e \in \Sigma_c) \wedge (g_e(d_q) = T) \wedge (\bar{q}_{e,d_q} \in \hat{W}^{k})\right) \right\}$
  - The set of all states such that $\omega^{G^s}(q) > k$ is $V^k = \{q \in (Q \setminus \bigcup_{j=0}^{k} W^j\}$. Remember that W^{k+1} is the set of states, such that the shortest failure-recovery path is $\omega^{G^s} = k+1$ transitions long. This means W^{k+1} can be rewritten to be the set of states in V^k, such that there is a controllable transition to a state $q' \in W^k$in the set W^k (in which for all states $\omega^{G^s}(q') = k$). So formally:

    $W^{k+1} = \left\{ q \in (Q \setminus \bigcup_{j=0}^{k} W^j) \;\middle|\; \exists e \in E \text{ s.t. } \left((o_e = l_q) \wedge (\sigma_e \in \Sigma_c) \wedge (g_e(d_q) = T) \wedge (\bar{q}_{e,d_q} \in W^{k})\right) \right\}$
  - Previously it was assumed that $W^k = \hat{W}^k$ is correct. It is shown that following this assumption, $W^{k+1} = \hat{W}^{k+1}$ is also correct. This proves that $W^k = \hat{W}^k$ is correct.

- Now all that's left is the failure-recovery distance for states q such that $k = \infty$. In Definition 7 this is the set of all states for which there is no failure-recovery path, let us define this set as W^∞. In Algorithm 2, the failure-recovery set $\hat{W}^k$ updates, till the final iteration $i = N$ is reached in which $\hat{W}^N = \emptyset$. This means that there are no states in $\hat{V}^N$ with a controllable transition to a failure-recoverable state. Thus, $\hat{V}^N$ represents all states in Q which there is no failure-recovery path, which shows that $W^\infty = \hat{V}^N$.

- Finally, we can see that $\omega^{G^s}(q) = \hat{\omega}(q)$ is correct for $k = 0$ on (line 5), for $k \in \mathbb{N}^+$ on (line 8) and for $k = \infty$ on (line 11).

Now that the correctness of $\hat{\omega}(q)$ is proven, it needs to be proven that the correct transitions are restricted. For $d \in D$ and transition $e \in E$, guard $g_e(d)$ is set to F if (and remains unaltered if not):

$(\bar{q}_{e,d} \notin Q_\lambda) \ \wedge \ (\omega^{G^s}(q_{e,d}) \leq \omega^{G^s}(\bar{q}_{e,d})) \wedge (\sigma_e \in \Sigma_c)$ following Definition 7

$(\bar{q}_{e,d} \notin Q_\lambda) \ \wedge \ (\hat{\omega}(q_{e,d}) \leq \hat{\omega}(\bar{q}_{e,d})) \wedge (\sigma_e \in \Sigma_c)$ following (line 12) in Algorithm 2

Because $\forall q \in Q,\ \omega^{G^s}(q) = \hat{\omega}(q)$, the correct guards are proven to be altered, and all others are proven to be unaltered. This proves that SSRCS(*G*) is a correct optimally-recovering supervisor of G^s. □

TU/e Technische Universiteit Eindhoven University of Technology

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

04-09-2019

Name

NICK PAAPE

ID-number

0782648

Signature

Submit the signed declaration to the student administration of your department.

[i] See: http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU