Taylor & Francis
Taylor & Francis Group

International Journal of Production Research

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tprs20

Modelling and control of manufacturing systems subject to context recognition and switching

Luiz Fernando Puttow Southier, Dalcimar Casanova, Luis Barbosa, Cesar Torrico, Marco Barbosa & Marcelo Teixeira

To cite this article: Luiz Fernando Puttow Southier, Dalcimar Casanova, Luis Barbosa, Cesar Torrico, Marco Barbosa & Marcelo Teixeira (2023) Modelling and control of manufacturing systems subject to context recognition and switching, International Journal of Production Research, 61:10, 3396-3414, DOI: 10.1080/00207543.2022.2081631

To link to this article: https://doi.org/10.1080/00207543.2022.2081631

INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH
2023, VOL. 61, NO. 10, 3396–3414
https://doi.org/10.1080/00207543.2022.2081631

Taylor & Francis
Taylor & Francis Group

Modelling and control of manufacturing systems subject to context recognition and switching

Luiz Fernando Puttow Southier[a], Dalcimar Casanova[b], Luis Barbosa[c], Cesar Torrico[b], Marco Barbosa[b] and Marcelo Teixeira [b]

[a]PPGIA, Pontifical Catholic University of Parana, Curitiba, Brazil; [b]Federal University of Technology Parana, Pato Branco, Brazil; [c]Universidade do Minho, Braga, Minho, Portugal

ABSTRACT

Finite-State Automata (FSA) are foundations for modelling, synthesis, verification, and implementation of controllers for manufacturing systems. However, FSA are limited to represent emerging features in manufacturing, such as the ability to recognise and switch contexts. One option is to enrich FSA with *parameters* that carry details about the manufacturing, which may favour design and control. A parameter can be embedded either on transitions or states of an FSA, and each approach defines its own modelling framework, so that their comparison and integration are not straightforward, and they may lead to different control solutions, modelled, processed and implemented distinctly. In this paper, we show how to combine advantages from parameters in manufacturing the modelling and control. We initially present a background that allows to understand each parameterisation strategy. Then, we introduce a conversion method that translates a design-friendly model into a synthesis-efficient structure. Finally, we use the converted models is synthesis, highlighting their advantages. Examples are used throughout the paper to illustrate and compare our results and tooling support is also provided.

ARTICLE HISTORY
Received 26 July 2021
Accepted 5 April 2022

KEYWORDS
Manufacturing systems; supervisory control; modelling integration; context handling; finite-state automaton

1. Introduction

Modern manufacturing systems aim to transform material into products by properly, flexibly and efficiently integrating people, equipment and technology (Esmaeilian, Behdad, and Wang 2016). It is expected that manufacturing components can interact with each other and with the environment in a concurrent manner, sharing resources and behaving in a safe, controllable and maximally permissive way (Hu, Liu, and Zhou 2015). In conjunction, those features make it hard the task of programming industrial controllers, as traditional paradigms for software development are usually inappropriate (Dotoli et al. 2017). Alternatives have been tested, for example, with simulation (Mourtzis 2020), optimisation (Pedrielli et al. 2018), and *intelligent* strategies (Guo et al. 2020; Hu and Liu 2015). Another option is to apply formal methods to exploit the *event* level of the system when expressing its behaviour and requirements (Rosa, Barbosa, and Teixeira 2019; Hu, Liu, and Yuan 2016). In this case, automated operations can be processed in order to calculate a controller that holds properties of interest.

When an industrial process is seen as a *Discrete Event System* (DES) (Cassandras and Lafortune 2009), the control objective is to obtain sequences of events to be allowed under control. Events are assumed to occur spontaneously in the system *plant*, and they are restricted by *specifications*, which are in general modelled using *Finite State Automata* (FSA). Then, formal approaches, such as the *Supervisory Control Theory* (SCT) (Ramadge and Wonham 1989), can be applied to synthesise controllers to be finally implemented in hardware (Qamsane, Tajer, and Philippot 2017).

Despite their practical relevance and formal background, FSA face significant limitations when modelling large and complex systems. Advanced features of flexible DES, such as context recognition and switching, are difficult to be expressed by ordinary FSA and they are usually associated with large state-spaces, which challenges both modelling and processing (Silva, Ribeiro, and Teixeira 2017). *Parameterized* FSA allow to address complexity issues in modelling and control of DESs. A so-called *parameter* is an engineered argument, embedded on a modelling formalism, that captures and carries

CONTACT Luiz Fernando Puttow Southier luizsouthier@gmail.com

context semantics throughout an FSA. This expands the information potential of a DES model and extends the related control techniques to cover a broader class of problems.

Technically, a parameter can be embedded either on transitions (Cury et al. 2015) or on states (Teixeira et al. 2015) of a DES model. In the first case, *Event-parameterised FSA* (EpFSA) are mechanisms that systematically map events into sets of new events, called *parameterized events*, that carry a given context semantic for the modelled DES. On the other hand, *State-parameterised FSA* (SpFSA) are variable-based models that can be designed by *Extended Finite State Automata* (Chen and Lin 2001), and control can be implement by disabling events based on the evaluation of logical formulas that manipulate variables.

In theory, both EpFSA and SpFSA play a similar role in modelling and control of DES, so that their choice should be straightforward. However, it does not exist so far in the literature an explicit way to compare them and expose their advantages. As each approach is structured within a specific framework, their integration is not direct and they may lead to different control solutions, modelled, computed and implemented distinctly (Teixeira et al. 2015; Cury et al. 2015; Rosa et al. 2017).

It has been reported that EpFSA benefit modelling and synthesis (Cury et al. 2015), besides to be modular (Teixeira, Cury, and de Queiroz 2018) and to have potential to reduce implementation costs (Rosa et al. 2017). But, in this case, the entire parametrization structure depends on an engineer to be constructed by hands. Differently, SpFSA are more suitable for modelling as they allow to express behaviour and control restrictions by manipulating simple formulas, instead of complex state-machines. However, variables of a SpFSA are natively atomic structures, which may limit modularisation, reducing the possibilities for taking advantages in the synthesis and implementation phases.

This paper shows how to combine advantages from EpFSA and SpFSA. We initially present a background that allows us to understand each parameterisation strategy, discussing their motivations and presenting examples. Then, we formally introduce a conversion method that translates a design-friendly model into a synthesis-efficient structure. Algorithms are provided to systematically extract meanings (parameters) from the states-space of a SpFSA, transferring them to an event-based EpFSA that expresses equivalent behaviour. Then, we show how the resulting EpFSA models can be exploited in modular synthesis, which leads to a set of controllers that tend to be obtained more efficiently with respect to the classical SpFSA-based synthesis. The impact of the resulting set of controllers on implementation issues is also discussed. We finally reproduce some classical results and examples from the literature, then we solve them using our method in order to evidence advantages. A tool (Southier 2021) is provided to support the algorithms presented in the paper.

The manuscript is structured as follows: the background about FSA, EpFSA and SpFSA is presented in Section 2; the conversion methods is introduced, discussed and exemplified in Section 3; synthesis aspects are exploited in Section 4; finally, Section 5 brings some conclusions and perspectives.

A first attempt to theoretically combine advantages from EpFSA and SpFSA was made (Southier et al. 2019). However, in that study, we do not consider: modular-synthesis advantages, modular conversion, tool implementation, and test cases. All these items feature in this paper.

2. Background

Many real world systems share the feature of being event-driven, i.e. their evolution in time is guided by the occurrence of asynchronous signals, called *events*, in opposition to time-driven behaviours. Systems that share these features are called *Discrete Event Systems* (DES) (Cassandras and Lafortune 2009) and they cover a wide range of domains, such as robotics, manufacturing, logistics, etc.

A DES can be modelled by *formal languages* (Hopcroft, Motwani, and Ullman 1939). *Events* define the basic structure of a language, and they are taken from a finite *alphabet* Σ, such that Σ^* denotes the set of all *strings* possibly built using events in Σ, including the *empty string* ε. Two strings $s, t \in \Sigma^*$ can be *concatenated* as st, and a subset $\mathcal{L} \subseteq \Sigma^*$ is said to be a *language*. The *prefix-closure* of $\mathcal{L} \subseteq \Sigma^*$ is $\overline{\mathcal{L}} = \{ s \in \Sigma^* \mid st \in \mathcal{L} \text{ for some } t \in \Sigma^* \}$.

In practice, when modelling a DES, it is usual to be interested in a class of languages called *regular*. A language is regular if it can be represented by a *Finite-State Automata* (FSA) (Cassandras and Lafortune 2009), which is a tuple $A = (\Sigma, Q, q^\circ, Q^\omega, \Gamma)$, where Σ is the alphabet; Q is the set of states; $q^\circ \in Q$ is the initial state; $Q^\omega \subseteq Q$ is the subset of marked states; and $\Gamma \subseteq Q \times \Sigma \times Q$ is the *transition relation*.

The transition between any two states $q_1, q_2 \in Q$ with the event $\sigma \in \Sigma$ is represented by $q_1 \xrightarrow{\sigma} q_2$, and it can be extended to a string $s \in \Sigma^*$ by $q_1 \xrightarrow{s} q_2$. Notation $q_1 \xrightarrow{\sigma}$ means $q_1 \xrightarrow{\sigma} q_2$ for some state q_2, and the same notation is used for strings in Σ^*, and $A \xrightarrow{s}$ means $q^\circ \xrightarrow{s} q$ for some state $q \in Q$, while $A \to q$ means $q^\circ \xrightarrow{s} q$ for some $s \in \Sigma^*$.

The set of events originating from q is given by $\Gamma(q) = \{ \sigma \in \Sigma \mid q \xrightarrow{\sigma} \}$, which can be restricted to a given subset

Σ' by $\Gamma^{\Sigma'}(q) = \{\sigma \in \Sigma' \subseteq \Sigma \mid q \xrightarrow{\sigma}\}$ and the enablement of an event σ from q is captured by $\Gamma^{\sigma}(q) = true$.

When a DES is modelled by an FSA A, its behaviour can be described by $\mathcal{L}(A) = \{s \in \Sigma^* \mid A \xrightarrow{s} q \in Q\}$, i.e. a language that includes all possible sequences generated by A. Its *marked* behaviour $\mathcal{L}^{\omega}(A) \subseteq \mathcal{L}(A)$, associated with the tasks accomplished by A, is defined as $\mathcal{L}^{\omega}(A) = \{s \in \Sigma^* \mid A \xrightarrow{s} q \in Q^{\omega}\}$. A language $\mathcal{L}(A)$ is said to be *non-blocking* if $\mathcal{L}(A) = (\mathcal{L}^{\omega}(A))$.

Two FSA, $A^1 = (\Sigma_1, Q_1, q_1^{\circ}, Q_1^{\omega}, \Gamma_1)$ and $A^2 = (\Sigma_2, Q_2, q_2^{\circ}, Q_2^{\omega}, \Gamma_2)$, can be composed as $A^1 \parallel A^2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, (q_1^{\circ}, q_2^{\circ}), Q_1^{\omega} \times Q_2^{\omega}, \Gamma_{A^1 \parallel A^2})$, where $\Gamma_{A^1 \parallel A^2}$ is defined as:

- $(q_1, q_2) \xrightarrow{\sigma} (q_1', q_2')$, if $\sigma \in \Sigma_1 \cap \Sigma_2$, $q_1 \xrightarrow{\sigma} q_1'$, and $q_2 \xrightarrow{\sigma} q_2'$;
- $(q_1, q_2) \xrightarrow{\sigma} (q_1', q_2)$, if $\sigma \in \Sigma_1 \setminus \Sigma_2$ and $q_1 \xrightarrow{\sigma} q_1'$;
- $(q_1, q_2) \xrightarrow{\sigma} (q_1, q_2')$, if $\sigma \in \Sigma_2 \setminus \Sigma_1$ and $q_2 \xrightarrow{\sigma} q_2'$;
- undefined, otherwise.

The operation $\parallel$ synchronises shared events and interleaves the others. Notation $A^{\parallel}$ means a composition of a set $\{A^1, \ldots A^n\}$ of FSA, i.e. $A^{\parallel} = A^1 \parallel \cdots \parallel A^n$.

2.1. Example of a manufacturing system

Consider the DES in Figure 1. Machines 1 and 2 receive external raw material (events a and c), manufacture workpieces types B and D, and deliver them to a buffer (events b and d) in arbitrary ordering. Machine 3 assembles pairs of workpieces type D in the buffer (event g), and it is disabled for other types. Machine 4 picks up workpieces from the buffer (event e), packs, and removes them from the system (event f).

Machines 1, 2, 3 and 4 can be modelled by the FSA G^1, G^2, G^3, and G^4 shown in Figure 2, so that the *plant* model is given by the composition $G = G^1 \parallel G^2 \parallel G^3 \parallel G^4$.

For control, it is assumed that the buffer has capacity of 2 workpieces, and one aims to prevent its overflow and underflow. In addition, machine 4 can only remove a pair type D from the buffer after they are paired by machine 3; otherwise (the buffer is not full or it includes only type B workpieces) machine 4 is enabled anytime and machine 3 is disabled. The *specification* $E = E^1 \parallel E^2 \parallel E^3$, shown

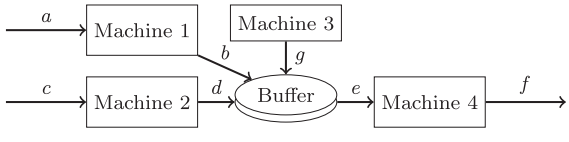


Figure 1. Manufacturing system with intermediate buffering.

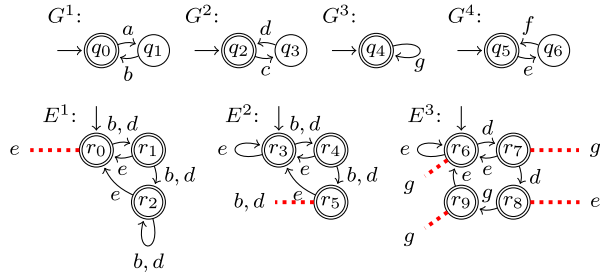


Figure 2. Plant and specification models for the example.

Table 1. Number of states (and transitions) for the example.

G	E	$K = E \parallel G$
8 (32)	7 (12)	56 (136)

in Figure 2, is modelled to control G. Dashed lines mean events disablement in E.

Model E^1 disables the event e when the buffer is empty (state r_0). Also, the events b and d are disabled by E^2 when the buffer is full (state r_5). Furthermore, E^3 disables e and enables g when the buffer has exactly two type D workpieces (state r_8), forcing in this case g to occur before e.

The composition $K = E \parallel G$ models the behaviour expected for G under the control of E and Table 1 shows the number of states of these models.

Model K can then be used as input to *synthesis* algorithms. This Section discusses only options for FSA-based DES modelling, while control synthesis frameworks are approached in Section 2.

2.2. A motivating gap of ordinary FSA models

Despite the recognised role ever played by FSA for DESs modelling, they face significant limitations when applied on real industry-scale problems. It can be shown Mohajerani, Malik, and Fabian (2017), Gohari and Wonham (2000), Cury et al. (2015), Teixeira et al. (2015), Rosa, Teixeira, and Malik (2018), and Teixeira, Cury, and de Queiroz (2018) that workflows commonly found in factory automation, such as recycling, buffering, and parallel manufacturing, may require hundreds of thousands of states to be expressed by ordinary FSA.

In Figure 1, for example, the activation of machine 3 depends on memorising certain sets of components in the buffer in order to ensure their correct manufacture. For a buffer with capacity of n workpieces, for instance, this modelling might require $n + 1$ states to be handled. If more than one type of workpiece were to be traced in the buffer, then this modelling could be far more complex.

As these are manual tasks, memorising complex sequences of events and states relies entirely on the

designer and it is not rarely unworkable. Alternatively, the literature (Teixeira et al. 2015; Cury et al. 2015) has suggested the possibility of enriching FSA with *parameters* that model extra *information* or *meaning* about specific parts of a DES behaviour. When properly engineered, a parameter allows identifying and isolating certain *contexts* from others, throughout the system model, and this can simplify design tasks. A formalism that supports embedding parameters into FSA is presented in the following.

2.3. Parameterisation of events

Event-parameterised FSA (EpFSA) are state-machines similar to ordinary FSA, but extended with a mechanism that stores parameters of a DES model in the form of *instances* of events, systematically mapped from the original event set. The result is a model that provides additional information of a DES, which in general benefits modelling.

Formally, a EpFSA maps each event $\sigma \in \Sigma$ into a set of *instances* $\Delta^\sigma = \{\delta_1, \delta_2, \ldots, \delta_n\}$, and δ_i aims to store certain context semantics, which is to be further defined. In this way, Σ becomes a *reference event set* for an *instantiated event set* $\Delta = \bigcup_{\sigma\in\Sigma} \Delta^\sigma$.

The mapping from Δ to the reference Σ can be implemented by $\Pi : \Delta^* \to \Sigma^*$, defined recursively such that $\Pi(\epsilon) = \epsilon$ and $\Pi(t\delta) = \Pi(t)\sigma$ for $t \in \Delta^*$, $\delta \in \Delta^\sigma$ and $\sigma \in \Sigma$ (Cury et al. 2015). This map can be generalised to any language $\mathcal{L}_\mathfrak{D} \subseteq \Delta^*$ by $\Pi(\mathcal{L}_\mathfrak{D}) = \{s \in \Sigma^* \mid \exists t \in \mathcal{L}_\mathfrak{D}, \Pi(t) = s\}$. Inversely, $\Pi^{-1} : \Sigma^* \to 2^{\Delta^*}$ maps from the reference set Σ to the dilated domain Δ, and it can be defined as $\Pi^{-1}(s) = \{t \in \Delta^* \mid \Pi(t) = s\}$, and extended to any language $\mathcal{L}$ by $\Pi^{-1}(\mathcal{L}) = \{t \in \Delta^* \mid \Pi(t) \in \mathcal{L}\}$.

For an FSA A, the map $\Pi^{-1}(A)$ means replacing the event of each transition by the respective set of instances $A_\mathfrak{D}$. Inversely, $\Pi(A_\mathfrak{D})$ recovers each original event, so that

$$\Pi(\Pi^{-1}(A)) = A \tag{1}$$

follows Cury et al. (2015). This process is illustrated in Figure 3.

When a DES plant is modelled by G, the corresponding EpFSA $G_\mathfrak{D} = \Pi^{-1}(G)$ represents G with enriched context semantics carried by instances of events, and the return to G is well defined from $\Pi(G_\mathfrak{D}) = G$ as in Equation (1). However, note that $G_\mathfrak{D}$ is a particular version of G that may enable more than one instance for each event enabled in G. In practice, this means that $G_\mathfrak{D}$ is capable of recognising different context semantics for G, but it is unable to choose which one applies at each step.

The choice for instances in $\Delta^\sigma = \{\delta_1, \delta_2, \ldots, \delta_n\}$ of an event $\sigma \in \Sigma$ depends on an additional model to *filter* $G_\mathfrak{D}$, assigning its appropriate semantics. A filter, denoted $H_\mathfrak{D}$, has the unique role of imposing *context switching* to the plant composition $G_\mathfrak{D} \| H_\mathfrak{D}$. That is, $H_\mathfrak{D}$ has no intention to restrict $G_\mathfrak{D}$ by disabling events completely, as specifications do. Instead, it simply chooses which instance of event should survive in $G_\mathfrak{D}$ when more than one are eligible. Furthermore, in this paper a filter is assumed to be *precise*, as follows.

Definition 2.1: For a DES plant G, let $G_\mathfrak{D} = \Pi^{-1}(G)$ and let $H_\mathfrak{D}$ be the filter for $G_\mathfrak{D}$. $H_\mathfrak{D}$ is *precise* if, for every state $q \in Q_{G_\mathfrak{D}\|H_\mathfrak{D}}$, and every event $\sigma \in \Sigma$, it follows that $|\Gamma^{\Delta^\sigma}(q)| \leq 1$.

Definition 2.1 ensures a single instance $\delta_i \in \Delta^\sigma$ is eligible upon a transition in $G_\mathfrak{D} \| H_\mathfrak{D}$. It represents the context to be enabled and all others are disabled.

2.3.1. Example with EpFSA

For the example in Section 2.1, the corresponding EpFSA are depicted in Figure 4.

Events b and d are instantiated such that $\Delta^b = \{b_0, b_1, b_2\}$ and $\Delta^d = \{d_0, d_1, d_2\}$ and they aim to carry extra information about the number of workpieces types B and D in the buffer, respectively. Events e and g are also parameterised as $\Delta^e = \{e_0, e_n, e_d\}$ and $\Delta^g = \{g_0, g_n, g_d\}$ such that: e_0 and g_0 mean that the buffer is empty; e_d and g_d represent a set of type D workpieces in the buffer; and e_n and g_n represent all other combinations.

Then, the plant models $G^i_\mathfrak{D}$, $i = 1, \ldots, 4$ enrich the corresponding plants G^i by recognising more contexts, but they are unable to choose which context applies at

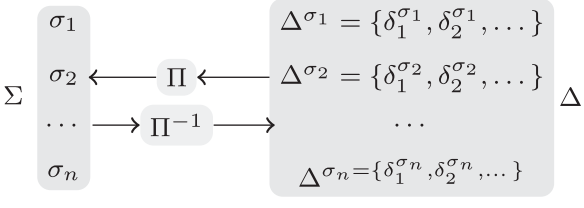


Figure 3. Event dilatation and recovering.

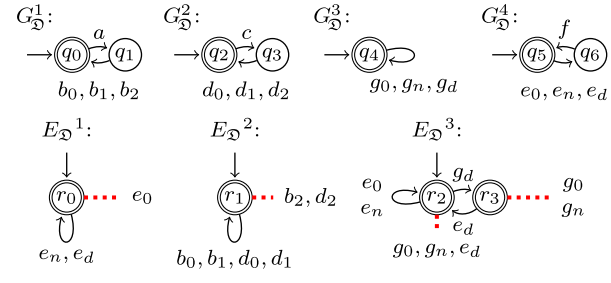


Figure 4. Plant and specification models for the example.

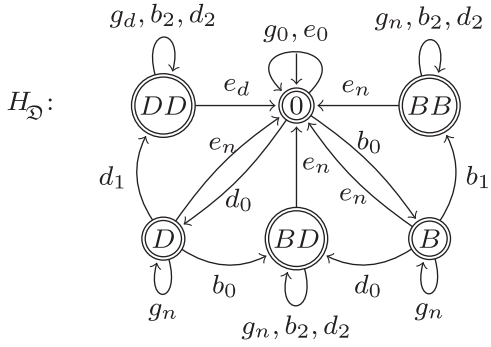


Figure 5. Filter for the example.

Table 2. Number of states (and transitions) for the example.

	Plant	Specification	Filter	Composition
FSA	8 (32)	7 (12)	–	56 (136)
EpFSA	8 (72)	2 (11)	6 (24)	56 (136)

every step of the plant. For example, transition $q_1 \xrightarrow{b_0,b_1,b_2} q_0$ is labelled with all instances of b, while it should precisely enable one. Such a complement is provided for the plant $G_{\mathfrak{D}}$ by $H_{\mathfrak{D}}$. For the plant in Figure 4, a precise $H_{\mathfrak{D}}$ can be designed as in Figure 5.

Note in Figure 5 that each state of $H_{\mathfrak{D}}$ enables at least one event in the instantiated event set $\Delta^{H_{\mathfrak{D}}} = \{b_i, d_i, e_j, g_j\}$, for $i = 0, 1, 2$ and $j = 0, n, d$, for each event in the corresponding reference set $\Pi(\Delta^{H_{\mathfrak{D}}}) = \{b, d, e, g\}$. This means that $H_{\mathfrak{D}}$ acts strictly as a complement for $G_{\mathfrak{D}}$, not ever restricting it. Furthermore, $H_{\mathfrak{D}}$ enables exactly one instance for each event in $\Pi(\Delta^{H_{\mathfrak{D}}})$ at each state, which means that it is precise. Therefore, when composed with $G_{\mathfrak{D}}$, $H_{\mathfrak{D}}$ leads to a plant that keeps a single string $t \in \mathcal{L}(G_{\mathfrak{D}} \parallel H_{\mathfrak{D}})$ for each corresponding $s \in \mathcal{L}(G)$, while still simplifying the specification $E_{\mathfrak{D}}$ with respect to E, besides to make it independent on buffer capacity.

In the example, overflow and underflow can be avoided by ${E_{\mathfrak{D}}}^1$ and ${E_{\mathfrak{D}}}^2$, and the correct actioning of Machine 3 is guaranteed by ${E_{\mathfrak{D}}}^3$. Table 2 shows the number of states of these models.

2.4. A motivating gap of EpFSA models

In comparison with FSA, EpFSA are more efficient to design and memorise contexts of a DES, at the price of modelling the filter $H_{\mathfrak{D}}$ to complement its plant model. Besides to be an additional step included in the design of DESs, modelling $H_{\mathfrak{D}}$ is also a manual task with complexity difficult to be estimated in advance.

Therefore, although the instantiated version of the DES model clearly benefits the project of specifications (as shown in the example of Section 2.3.1 and in the literature (Cury et al. 2015; Teixeira, Cury, and de Queiroz 2018)), it in contrast transfers modelling effort to the plant, reverting parts of its advantages and making its usefulness at least questionable.

In the following, we show that the filter $H_{\mathfrak{D}}$ can nevertheless be derived from a different notion of modelling, which tends to be more tunned with the designer's perception, alleviating the burden added to the plant modelling by EpFSA. We also exploit the automatic construction of a modular version of $H_{\mathfrak{D}} = H_{\mathfrak{D}1} \parallel \cdots \parallel H_{\mathfrak{D}m}$, which can further benefit other steps of the project of controllers, such as control synthesis (see Section 4), in addition to design issues.

2.5. Parameterisation of states

State-parameterised FSA (SpFSA) are similar to ordinary FSA, but their transitions include *formulas* over *variables*. Formally, a SpFSA can be expressed as a tuple $A_{\mathfrak{E}} = (\Sigma, V, Q, q^{\circ}, Q^{\omega}, P_V, \Gamma)$ (Chen and Lin 2001; Teixeira et al. 2015) where Σ is the alphabet of events; $V = \{v_1, \ldots, v_n\}$ is the set of variables; Q is the set of states[1]; $q^{\circ} \in Q$ is the initial state; $Q^{\omega} \subseteq Q$ is the subset of marked states; P_V is the set of formulas over V; and $\Gamma \subseteq Q \times \Sigma \times P_V \times Q$ is the transition set that leads from a state in Q to another, with an event taken from Σ, and formulas taken from the set of formulas P_V.

In this definition, a variable v is an entity with a finite domain $Dom(v)$ and an initial value $v^o \in Dom(v)$. Then, $V = \{v_1, \ldots, v_n\}$ has a domain $Dom(V) = Dom(v_1) \times \cdots \times Dom(v_n)$. In order to manipulate variables, SpFSA use formulas from $P_V = \{p_1, \ldots, p_m\}$. In conjunction, variables and formulas establish a new transition mechanism, also denoted differently (Chen and Lin 2001): a transition from q_0 to q_1 in Q, with the event $\sigma \in \Sigma$ and formula $p \in P_V$,is exposed as $q_0 \xrightarrow{\sigma:p} q_1$. Notation $q_0 \xrightarrow{\sigma:p}$ means $q_0 \xrightarrow{\sigma:p} q_1$ for some state $q_1 \in Q$.

In order to differentiate variable values before and after the transition, a *next-state* variable set $V' = \{v'_1, \ldots, v'_n\}$, with $Dom(V) = Dom(V')$, is associated to V. Then, $\bar{v} \in Dom(v)$ and $\bar{v}' \in Dom(v')$ mean the value assumed by the variable v respectively in the current and next-state.

Structurally, each formula $p \in P_V$ is constructed using a set of current-state variables, denoted $\mathcal{V}_p$, and a set of next-state variables, $\mathcal{V}'_p$, such that $\mathcal{V}_p \subseteq V$ and $\mathcal{V}'_p \subseteq V'$. Then, the effect of evaluating formulas upon transitions can be described by their impact on variable values associated with current and next-states. This can be captured by the variation imposed to the generated tuple $\hat{v}'_p = (\bar{v}_i', \ldots, \bar{v}_j') \in Dom(\mathcal{V}'_p) = Dom(v'_i) \times \cdots \times Dom(v'_j)$ with respect to the source tuple $\hat{v}_p =$

$(\overline{v_n}, \ldots, \overline{v_m}) \in Dom(\mathcal{V}_p) = Dom(v_n) \times \cdots \times Dom(v_m)$, which are called *valuations*.

A formula $p \in P_V$ can either *update* or *test* variable values upon transitions. An update p replaces the valuation $\hat{v}_p$, associated to the current-state, to a new valuation $\hat{v}'_p$ in the reached state, which is denoted by $\hat{v}'_p = p(\hat{v}_p)$. A valuation $\hat{v}_p$ is said to be *valid* for p if $\hat{v}_p \in Dom(\mathcal{V}_p)$, and $\hat{v}'_p \in Dom(\mathcal{V}'_p)$. Differently, test formulas (or *guards*) do not change any variable value ($\mathcal{V}'_p = \emptyset$), they simply test values associated to the current state ($\hat{v}_p$), disabling or not the transition depending on the test result, i.e. $p(\hat{v}_p) = true$ or $p(\hat{v}_p) = false$. Therefore, any valuation $\hat{v}_p \in Dom(\mathcal{V}_p)$ is valid over test formulas.

For the results in this paper, it is sometimes necessary to infer about some particular variable values inside a larger tuple. Let $V_1 = \{v_i, \ldots, v_j\}$ be a set of variables and $\hat{v}_1 = (\overline{v_i}, \ldots, \overline{v_j})$ be a valuation on V_1. We denote by $\hat{v}_1(V_2)$ the values in $\hat{v}_1$ that correspond to variables in V_2, for any $V_2 \subseteq V_1$.

To exemplify, let x and y be two variables such that $Dom(x) = Dom(y) = \{1, 2, 3\}$, and $V = \{x, y\}$. Now let $x' = x + y + 1$ be a formula $p_1 \in P_V$ that changes the value of x', with $\mathcal{V}_{p_1} = \{x, y\}$, $\mathcal{V}'_{p_1} = \{x\}$, $Dom(\mathcal{V}_{p_1}) = Dom(V)$, and $Dom(\mathcal{V}'_{p_1}) = Dom(x)$. The current-state valuations $\hat{v}_{p_1}$ are in the form of $(\bar{x}, \bar{y})$ and the next-state valuations $\hat{v}'_{p_1}$ are in the form of $(\bar{x})$.

Then, for a valuation $\hat{v}_{p_1} = (1, 1)$, it follows that $p_1(\hat{v}_{p_1}) = (3)$ and $\hat{v}_{p_1} = (1, 1)$ is valid with respect to p_1. For $\hat{v}_{p_1} = (2, 2)$, then $p_1(\hat{v}_{p_1}) = (5) \notin Dom(\mathcal{V}'_{p_1})$, and $\hat{v}_{p_1} = (2, 2)$ is not valid for p_1. For a test formula, for example $y > 2$, it follows that $\mathcal{V}_{p_2} = \{y\}$, $Dom(\mathcal{V}_{p_2}) = Dom(y)$, and the valuations $\hat{v}_{p_2}$ are in the form of $(\bar{y})$. Then, $p_2(\hat{v}_{p_2})$ is *true* for the valuation $\hat{v}_{p_2} = (3)$, and *false*, otherwise.

Next, plant SpFSA are assumed to implement only updates. This is because we want updates to be context selectors for the plant, without imposing any restriction. Differently, specifications only test value combinations, from those enabled by the plant, and have the purpose of disabling transitions upon false evaluations. These assumptions are necessary conditions for the synthesis algorithms with SpFSA to hold (Teixeira et al. 2015; Malik and Teixeira 2016, 2020, 2021).

Furthermore, in order for the context selection to be precise, i.e. for it to enable a single value combination at each state, updates are assumed to be *exact*, i.e. for each valuation $\hat{v}_p$, an update $p(\hat{v}_p)$ leads to a unique valuation $\hat{v}'_p$. This differs from approaches that also handle variable abstraction, which in general deal with nondeterminism of variable values (Teixeira et al. 2015; Malik and Teixeira 2020). Finally, updates are all required to be *convergent*, i.e. for any two transitions $x_1 \xrightarrow{\sigma:p_1} y_1$ and $x_2 \xrightarrow{\sigma:p_2} y_2$ with the same event $\sigma \in \Sigma$, it holds that $\hat{v}'_{p_1} = \hat{v}'_{p_2}$. This means that two updates cannot implement divergent changes on a variable, upon a same event.

Remark that, so far, a SpFSA is exposed in its *implicit* form, i.e. including all formulations. In this case, the value combination to be associated with each state is unknown until each formula is in fact evaluated. After that, the SpFSA becomes *explicit*, as each variable value is revealed in each state and the state-space is unfolded. In this paper, we use unfolded SpFSA for quantifications, such as state-space, which is a measure for computational effort.

Given a SpFSA $A_v = (\Sigma, V, Q, q^\circ, Q^\omega, P_V, \Gamma)$, the *explicit version* of a transition $q_0 \xrightarrow{\sigma:p} q_1$ is written as $(q_0, \hat{v}_p) \xrightarrow{\sigma} (q_1, \hat{v}'_p)$, such that $\hat{v}_p$ and $\hat{v}'_p$ are the valuations in q_0 and $q_1{}'$, respectively. This can be extended to strings $s \in \Sigma^*$ by $(q_0, \hat{v}_{p_0}) \xrightarrow{s} (q_1, \hat{v}'_{p_1})$. Notation $(q_0, \hat{v}_p) \xrightarrow{\sigma}$ means $(q_0, \hat{v}_p) \xrightarrow{\sigma} (q_1, \hat{v}'_p)$, i.e. $\hat{v}_p$ in q_0 enables the event σ leading to some state $q_1 \in Q$ and some next-state valuation $\hat{v}'_p$. The same notation is used for strings in Σ^*, where $A_v \xrightarrow{s}$ means $(q^\circ, \hat{v}_{p_0}) \xrightarrow{s} (q, \hat{v}'_{p_1})$ for some state $q \in Q$ and some next-state valuation $\hat{v}'_{p_1}$, while $A \to q$ means $(q^\circ, \hat{v}_{p_0}) \xrightarrow{s} (q, \hat{v}'_{p_1})$ for some $s \in \Sigma^*$. Then, $\mathcal{L}(A_v) = \{ s \in \Sigma^* \mid A_v \xrightarrow{s} q \in Q \}$ is the language of A_v.

2.5.1. *Example with* SpFSA

Here, we show how the example presented in Section 2.1 can be remodelled using SpFSA and we highlight possible benefits of that.

Let the FSA G^1, G^2, G^3 and G^4 from Figure 2 be now modelled respectively by the SpFSA $G^1_{\mathfrak{E}}$, $G^2_{\mathfrak{E}}$, $G^3_{\mathfrak{E}}$ and $G^4_{\mathfrak{E}}$ as in Figure 6. Structurally, they are essentially the same, except that the SpFSA update variables $x, y \in V$. They are defined with domains $Dom(x) = Dom(y) = \{0, 1, 2, 3\}$ and initial values $x^o = y^o = 0$, in order to memorise the number of workpieces inserted in the buffer by Machines 1 and 2, respectively. The variables are updated by formulas on transitions with the events b and d (insertion) and e (removal).

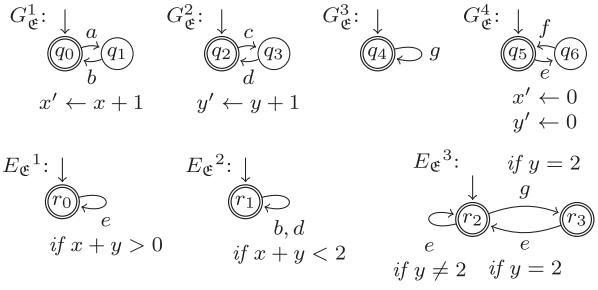


Figure 6. SpFSA models for the example.

Table 3. Number of states (and transitions) for the example.

	Plant	Specification	Filter	Composition
FSA	8 (32)	7 (12)	-	56 (136)
EpFSA	8 (72)	2 (11)	6 (24)	56 (136)
SpFSA	72 (380)	2 (14)	-	56 (136)

The benefits brought by updating x and y in the system plant can be really seen when remodelling E. Underflow and overflow can now be prevented respectively by the modular specifications $E_{\mathfrak{E}}{}^1$ and $E_{\mathfrak{E}}{}^2$, while the correct activation of machine 3 (occurrence of g only for a pair type D) is controlled by the specification $E_{\mathfrak{E}}{}^3$, depicted in Figure 6.

Model $E_{\mathfrak{E}}{}^i$ tests the values of x and y before enabling the events b, d, e and g, which is independent on buffer capacity. This comes at the price of constructing the updates in the plant model, but this is rather an intuitive task, in general.

For SpFSA, the plant model is given by the composition $G_{\mathfrak{E}} = G_{\mathfrak{E}}^1 \parallel G_{\mathfrak{E}}^2 \parallel G_{\mathfrak{E}}^3 \parallel G_{\mathfrak{E}}^4$, and the specification is modelled by $E_{\mathfrak{E}} = E_{\mathfrak{E}}{}^1 \parallel E_{\mathfrak{E}}{}^2 \parallel E_{\mathfrak{E}}{}^3 \parallel E_{\mathfrak{E}}{}^4$, so that $K_{\mathfrak{E}} = G_{\mathfrak{E}} \parallel E_{\mathfrak{E}}$ expresses the behaviour expected from $G_{\mathfrak{E}}$ when controlled by $E_{\mathfrak{E}}$. Table 3 shows the number of states and transitions for these models. For the sake of clarity, it also repeats the statistics for all approaches.

Note that $K_{\mathfrak{E}}$ is modelled with 56 unfolded states, therefore the same number as K (and $K_{\mathfrak{D}}$). This suggests that the computational cost to process both models is the same, but the modelling of $K_{\mathfrak{E}}$ and $K_{\mathfrak{D}}$ is much simpler and it remains simple for any buffer size to be considered.

Actually, the difference between $K_{\mathfrak{E}}$ and $K_{\mathfrak{D}}$ is that $K_{\mathfrak{E}}$ can keep *memory* of a DES model by manipulating simple formulas, engineered over a high-level view of the system. Differently, obtaining $K_{\mathfrak{D}}$ requires the manual construction of a filter $H_{\mathfrak{D}}$, which is unsystematic and it can be complex (see Figure 5). In practice, the construction of $K_{\mathfrak{D}}$ only transfers modelling effort from the specification to the filter, so that its real advantages are unclear to be claimed.

2.5.2. *A motivating gap of* SpFSA *models*

Despite possible modelling advantages brought by the easy way behaviours and restrictions are expressed, SpFSA do not fully exploit modularisation under control synthesis. There are abstraction techniques (Teixeira et al. 2015) that work modularly (Malik and Teixeira 2016, 2020) by removing unnecessary variables from synthesis. Variables to be ruled out are chosen according to the role they play in control, and this reduces the computational effort needed to process synthesis. However, those techniques do not work with partial abstractions, i.e. abstractions that, besides removing unnecessary variables, also remove unnecessary parts of necessary variables.

Partial abstractions are more complicate to be constructed, as variable domains are intrinsically inseparable, so that they are usually taken entirely in synthesis. When a variable domain is large and combined with other variable domains, it tends to create huge states-spaces that limit algorithmic treatment and prevent, to some extent, modelling advantages to propagate through synthesis and implementation phases. Parallel advantages can be taken by converting SpFSA into EpFSA, as shown in the following.

3. Proposed conversion method

This Section shows how EpFSA can be derived from SpFSA. The idea is to conduct modelling using SpFSA, converting then into EpFSA for synthesis. Advantages of this method are quantified in Section 4, and the general context of the conversion is illustrated in Figure 7.

The first column illustrates the usual case where a DES and its specifications are respectively modelled by the FSA G and E, where $K = G \parallel E$ is the synthesis input using the classic monolithic SCT framework (Ramadge and Wonham 1989).

In the third column, SpFSA are used to handle modelling. It uses the same event set Σ as for FSA, but it is helped in modelling by variables and updates. The result is that FSA G and E are now expressed by SpFSA $G_{\mathfrak{E}}$ and $E_{\mathfrak{E}}$, which leads to a composition $K_{\mathfrak{E}}$ that can be used as synthesis input for the algorithm in Teixeira et al. (2015).

The second column structures modelling on a different, dilated, set of events, Δ. This leads to a plant $G_{\mathfrak{D}}$ that is complemented with $H_{\mathfrak{D}}$ for context recognition and switching. In the same way, E turns to be modelled by $E_{\mathfrak{D}}$ and the synthesis input is then given by the composition $K_{\mathfrak{D}} = G_{\mathfrak{D}} \parallel E_{\mathfrak{D}} \parallel H_{\mathfrak{D}}$, such that $\Pi(K_{\mathfrak{D}})$ and K are expected to be language equivalent[2] (Cury et al. 2015).

Summarizing, FSA, SpFSA and EpFSA are expected to compose equivalent synthesis inputs. However, ordinary FSA are usually limited for modelling. Differently, SpFSA are more suitable for modelling, but not directly for synthesis.

For example, the algorithm in Teixeira et al. (2015) synthesises for controllability and least restrictiveness of SpFSA, but it does not work for nonblocking results. Furthermore, a supervisor computed from SpFSA is another SpFSA that makes implementation more difficult, as it still includes formulas and updates. Finally, the SpFSA synthesis result is also difficult to be compared with an ordinary supervisor, as SpFSA and FSA do not usually share the same properties (Malik and Teixeira 2020).

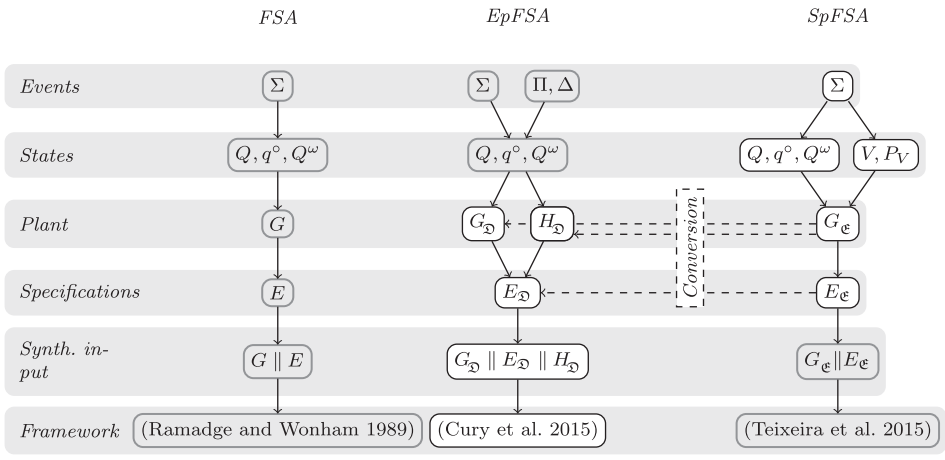


Figure 7. Structural comparison of FSA, SpFSA and EpFSA.

This motivates us to take advantages from SpFSA in modelling, but using their ordinary EpFSA version, obtained from our conversion method, for the further steps of control engineering. EpFSA have the advantage of keeping modularity by renaming events according to contexts, which in synthesis is essential to split computation in smaller, simpler, parts (Teixeira, Cury, and de Queiroz 2018). Next, we present the proposed conversion method in details.

3.1. *Converting* SpFSA *into* EpFSA

Recall that, in a SpFSA, a transition with a given event $\sigma \in \Sigma$ may implement formulas $p \in P_V$ that use or test variables values $(\overline{v}_i, \ldots, \overline{v}_j) = \hat{v}_p \in Dom(\mathcal{V}_p)$, leading to a deterministic valuation $(\overline{v}_k', \ldots, \overline{v}_l') = \hat{v}_p' \in Dom(\mathcal{V}_p')$ at every state.

Variables in the sets $\mathcal{V}_p$ and $\mathcal{V}_p'$, and their valuations $\hat{v}_p$ and $\hat{v}_p'$, are parameters that represent the DES context in its current and next-states, respectively. In order to reproduce similar effect using parameterised events, one option is concatenating each original event to a parameters describing the change from current to next context. This notion leads to a new set of parameterised events.

Systematically, if there is no context switching upon a transition, i.e. the transition does not implement any formula and $\mathcal{V}_p \cup \mathcal{V}_p' = \emptyset$, then there is no need for creating a parameterised event. However, if $\mathcal{V}_p \cup \mathcal{V}_p' \neq \emptyset$, then at least one variable value has been modified or tested upon the transition and parameters representing this modification must be added to the in-construction EpFSA. This construction is formalised by first defining a set of variables $\mathbb{V}^\sigma = \bigcup_j \mathcal{V}_{p_j} \cup \mathcal{V}_{p_j}'$, for all $\xrightarrow{\sigma:p_j}$. Then, for a valuation $\hat{\mathbb{V}}_i^\sigma \in \hat{\mathbb{V}}^\sigma$, such that $\hat{\mathbb{V}}^\sigma = Dom(\mathbb{V}^\sigma)$, the context switching can be captured by the parameterised event $\sigma_{\mathbb{V}^\sigma \hat{\mathbb{V}}_i^\sigma}$, meaning that the variables in $\mathbb{V}^\sigma$ have the valuation $\hat{\mathbb{V}}_i^\sigma$ before the transition with σ.

For instance, consider the transitions $\xrightarrow{\sigma:w' \leftarrow 0}$ and $\xrightarrow{\sigma:z' \leftarrow w}$ such that $Dom(z) = Dom(w) = \{0, 1\}$. First, $\mathbb{V}^\sigma = \{w, z\}$ is created, because $\mathcal{V}_{w' \leftarrow 0} = \{w\}$, $\mathcal{V}'_{w' \leftarrow 0} = \emptyset$, $\mathcal{V}_{z' \leftarrow w} = \{z\}$, and $\mathcal{V}'_{z' \leftarrow w} = \{w\}$. Observe that all variables involved in the updates related to σ are elements in $\mathbb{V}^\sigma$. Then, $\hat{\mathbb{V}}^\sigma = Dom(\mathbb{V}^\sigma) = \{(0,0), (0,1), (1,0), (1,1)\}$ includes all valid combinations for variables in $\mathbb{V}^\sigma$. Now, by concatenating the event σ, the variables $\mathbb{V}^\sigma$, and each possible combination $\hat{\mathbb{V}}_i^\sigma \in \hat{\mathbb{V}}^\sigma$, one creates the parameterised events as $\sigma_{\mathbb{V}^\sigma \hat{\mathbb{V}}_i^\sigma}$, i.e. σ_{wz00}, σ_{wz01}, σ_{wz10}, and σ_{wz11}.

By doing this for all events, and all possible variable changes, an EpFSA can preserve exactly the same semantic of updates as a SpFSA, but using a different mechanism, which enriches transitions instead of states. In this paper, the entire conversion is founded on the four-steps procedure in Figure 8.

The first step extracts all contexts to be possibly assumed by the SpFSA plant $G_{\mathfrak{D}}$ and specification

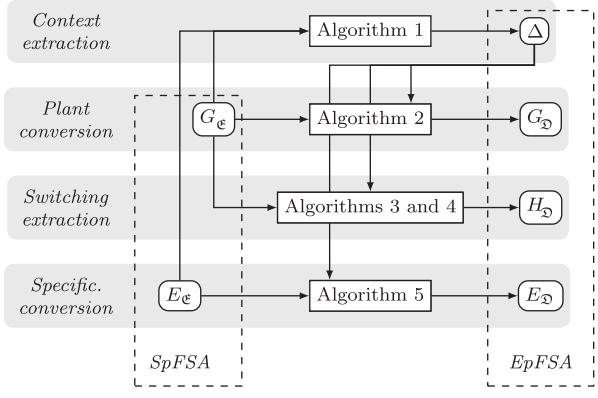


Figure 8. Conversion process from SpFSA to EpFSA.

Algorithm 1: Context extraction from SpFSA to EpFSA

```
input : plant G_𝔈 = (Σ_G, V_G, Q_G, q°_G, Q^ω_G, P_VG, Γ_G)
        specification E_𝔈 = (Σ_E, V_E, Q_E, q°_E, Q^ω_E, P_VE, Γ_E)
output: set of parameterized events Δ
1  begin
2      Δ ← ∅
3      foreach σ ∈ Σ_G do
4          Δ^σ ← ∅, 𝕍^σ ← ∅, 𝕍̂^σ ← ∅
5          foreach transition q1 --σ:p--> q2 ∈ Γ_G ∪ Γ_E, such that
             q1, q2 ∈ Q_G ∪ Q_E, p ∈ P_VG ∪ P_VE and V_p ∪ V'_p ≠ ∅
             do
6              𝕍^σ ← 𝕍^σ ∪ V_p ∪ V'_p
7          end
8          foreach 𝕍̂^σ_i ∈ Dom(𝕍^σ), such that 𝕍^σ = {v_j, ..., v_k}
             and Dom(𝕍^σ) = Dom(v_j) × ··· × Dom(v_k) do
9              𝕍̂^σ ← 𝕍̂^σ ∪ 𝕍̂^σ_i, such that 𝕍̂^σ_i (V_p ∪ V'_p) is valid
                 with respect to p ∀ transitions
                 q1 --σ:p--> q2 ∈ Γ_G ∪ Γ_E
10         end
11         Δ^σ ← Δ^σ ∪ {σ_{𝕍^σ 𝕍̂^σ_i}}, for each 𝕍̂^σ_i ∈ 𝕍̂^σ
12         if 𝕍^σ = ∅ then
13             Δ^σ ← σ
14         end
15         Δ ← Δ ∪ Δ^σ
16     end
17     return Δ
18 end
```

$E_{\mathfrak{E}}$ (Algorithm 1). This generates a set of parameterised events Δ, each representing a context. Then, the SpFSA plant $G_{\mathfrak{E}}$ is converted into an EpFSA plant $G_{\mathfrak{D}}$ (Algorithm 2). As initially $G_{\mathfrak{E}}$ may not be precise, a filter is constructed in the third step by extracting the context switching behaviour from variable updates (Algorithms 3 and 4). Finally, the fourth step maps the restrictions modelled by the specification, from SpFSA to a EpFSA (Algorithm 5). Each algorithm is introduced, discussed and exemplified in details in the following.

3.1.1. Context extraction

For context extraction, Algorithm 1 initially takes the new alphabet Δ as empty (line 2). Then, for each event $\sigma \in \Sigma$, it constructs a set of parameterised events Δ^σ representing all possible contexts that σ can assume. $\mathbb{V}^\sigma$ is the set of variables involved on context representation of σ and $\hat{\mathbb{V}}^\sigma$ is the set of possible valuations for $\mathbb{V}^\sigma$, that is, $\hat{\mathbb{V}}^\sigma_i \in \hat{\mathbb{V}}^\sigma$ such that $\hat{\mathbb{V}}^\sigma_i \in Dom(\mathbb{V}^\sigma)$.

Initially, Δ^σ, $\mathbb{V}^\sigma$ and $\hat{\mathbb{V}}^\sigma$ are all empty (line 4). Then, all transitions labelled $\sigma : p$ are read from the input SpFSA $G_{\mathfrak{E}}$ and $E_{\mathfrak{E}}$ (line 5) in order to identify the variables that represent contexts for σ. Variables in $\mathcal{V}_p \cup \mathcal{V}'_p$ are added to the set $\mathbb{V}^\sigma$. Next, each valuation $\hat{\mathbb{V}}^\sigma_i \in Dom(\mathbb{V}^\sigma)$ is inserted into the set $\hat{\mathbb{V}}^\sigma$, as long as $\hat{\mathbb{V}}^\sigma_i$ is valid with respect to all formulas on transitions with σ (line 8).

Finally, each parameterised event is created by associating to σ the variables in $\mathbb{V}^\sigma$ and each of the valid valuations $\hat{\mathbb{V}}^\sigma_i$ in $\hat{\mathbb{V}}^\sigma$. If no parameterised event have to be created (i.e. $\mathbb{V}^\sigma = \emptyset$) then σ is added to Δ^σ. As a result, Algorithm 1 identifies all possible contexts in $G_{\mathfrak{E}}$ and $E_{\mathfrak{E}}$, and reproduces them in Δ by using a mechanism that is free from formulas and variables.

3.1.1.1. Example. For the input models in Figure 6, it is possible to obtain the corresponding $\Delta_{\mathfrak{D}}$ by using Algorithm 1. Events a, c and f have no transitions with formulas in Figure 6 ($\mathcal{V}_p = \mathcal{V}'_p = \emptyset$ follows for all transitions), then $\Delta^a = \{a\}$, $\Delta^c = \{c\}$, and $\Delta^f = \{f\}$. The formulas $x' \leftarrow x + 1$ and *if* $x + y < 2$ are associated with the event b, then the set of variables associated with b is $\mathbb{V}^b = \{x, y\}$. The set of valuations in the form (x, y) of $\mathbb{V}^b$ that are valid with respect to all formulas associated to b in the plant is $\hat{\mathbb{V}}^b = \{(0,0), (0,1), (0,2), (0,3), (1,0), (1,1), (1,2), (1,3), (2,0), (2,1), (2,2), (2,3)\}$. Because $\bar{x} == 3$ would assign a value beyond the domain of x, the valuations $(3,0)$, $(3,1)$, $(3,2)$ and $(3,3)$ are not valid. Therefore, the set of parameterised events for b is $\Delta^b = \{b_{xy00}, b_{xy01}, b_{xy02}, b_{xy03}, b_{xy10}, b_{xy11}, b_{xy12}, b_{xy13}, b_{xy20}, b_{xy21}, b_{xy22}, b_{xy23}\}$.

In the same way, for the event d, $\mathbb{V}^d = \{x, y\}$, $\hat{\mathbb{V}}^d = \{(0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2), (3,0), (3,1), (3,2)\}$, and $\Delta^d = \{d_{xy00}, d_{xy01}, d_{xy02}, d_{xy10}, d_{xy11}, d_{xy12}, d_{xy20}, d_{xy21}, d_{xy22}, d_{xy30}, d_{xy31}, d_{xy32}\}$; for e, $\mathbb{V}^e = \{x, y\}$, $\hat{\mathbb{V}}^e = \{(0,0), (0,1), (0,2), (0,3), (1,0), (1,1), (1,2), (1,3), (2,0), (2,1), (2,2), (2,3), (3,0), (3,1), (3,2), (3,3)\}$, and $\Delta^e = \{e_{xy00}, e_{xy01}, e_{xy02}, e_{xy03}, e_{xy10}, e_{xy11}, e_{xy12}, e_{xy13}, e_{xy20}, e_{xy21}, e_{xy22}, e_{xy23}, e_{xy30}, e_{xy31}, e_{xy32}, e_{xy33}\}$; and, for g, $\mathbb{V}^g = \{y\}$, $\hat{\mathbb{V}}^g = \{(0), (1), (2), (3)\}$, and $\Delta^g = \{g_{y0}, g_{y1}, g_{y2}, g_{y3}\}$.

3.1.2. Plant conversion

In order to convert the plant into an EpFSA, Algorithm 2 replaces each event σ of a transition in $G^i_{\mathfrak{E}}$ by the corresponding parameterised set of events $\Delta^\sigma \in \Delta$. For that, it initially constructs a structure of states identical to the input SpFSA. As the new alphabet and transition relation are undefined at this point, they are started as empty.

For each transition labelled with an event σ in the input SpFSA $G^i_{\mathfrak{E}}$ (line 4), the corresponding events $\Delta^\sigma \in \Delta$ are included into the output EpFSA alphabet $\Delta_{\mathfrak{D}}$ (line 5). Then, new transitions are created replacing the event σ by each of the parameterised events in Δ^σ (line 6). Finally, those transitions are included on set $\Gamma_{\mathfrak{D}}$.

3.1.2.1. Example. By taking the alphabet Δ, generated by Algorithm 1, it is possible to create the EpFSA in Figure 9 by providing the SpFSA plants in Figure 6 as

Algorithm 2: Plant conversion from SpFSA to EpFSA

input : SpFSA plant $G^i_{\mathfrak{E}} = (\Sigma, V, Q, q^\circ, Q^\omega, P_V, \Gamma)$
set of parameterized events Δ
output: EpFSA plant $G^i_{\mathfrak{D}} = (\Delta_{\mathfrak{D}}, Q_{\mathfrak{D}}, q^\circ_{\mathfrak{D}}, Q^\omega_{\mathfrak{D}}, \Gamma_{\mathfrak{D}})$

1 **begin**
2 $Q_{\mathfrak{D}} \leftarrow Q$, $Q^\omega_{\mathfrak{D}} \leftarrow Q^\omega$, $\Delta_{\mathfrak{D}} \leftarrow \emptyset$, $\Gamma_{\mathfrak{D}} \leftarrow \emptyset$
3 $q^\circ_{\mathfrak{D}} \leftarrow q^\circ$
4 **foreach** *transition* $q_1 \xrightarrow{\sigma:p} q_2 \in \Gamma$, *such that* $q_1, q_2 \in Q$, $\sigma \in \Sigma$ *and* $p \in P_V$ **do**
5 $\Delta_{\mathfrak{D}} \leftarrow \Delta_{\mathfrak{D}} \cup \Delta^\sigma$
6 $\Gamma_{\mathfrak{D}} \leftarrow \Gamma_{\mathfrak{D}} \cup \{q_1 \xrightarrow{\delta} q_2\}$, for all $\delta \in \Delta^\sigma$
7 **end**
8 **return** $G^i_{\mathfrak{D}}$
9 **end**

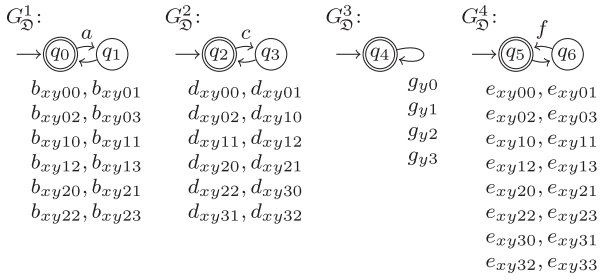


Figure 9. Converted plant models.

Algorithm 3: Switching extraction from SpFSA to EpFSA

input : plant model $G_{\mathfrak{E}} = (\Sigma, V, Q, q^\circ, Q^\omega, P_V, \Gamma)$
set of parameterized events Δ
output: set H' of non-modular EpFSA representing the plant context switching

1 **begin**
2 $H' \leftarrow \emptyset$
3 **foreach** *variable* $v_i \in V$, *such that* $Dom(v_i) = \{\overline{v}_{i1}, \ldots, \overline{v}_{in}\}$ *and* $v^o_i \in Dom(v_i)$ **do**
4 $Q_{v_i} \leftarrow \{q_{\overline{v}_{i1}}, \ldots, q_{\overline{v}_{in}}\}$
5 $q^\circ_{v_i} \leftarrow q_{v^o_i}$, $Q^\omega_{v_i} \leftarrow Q_{v_i}$
6 $\Delta^{v_i} \leftarrow \emptyset$, $\Gamma_{v_i} \leftarrow \emptyset$
7 **foreach** *transition* $q \xrightarrow{\sigma:p} \in \Gamma$ **do**
8 **foreach** *parameterized event* $\sigma_{\mathbb{V}^\sigma \hat{\mathbb{V}}^\sigma} \in \Delta^\sigma$, *such that* $v_i \in \mathbb{V}^\sigma$ **do**
9 $curr \leftarrow \hat{\mathbb{V}}^\sigma(v_i)$
10 **if** $v_i \in \mathcal{V}'_p$ **then**
11 $next \leftarrow p(\hat{\mathbb{V}}^\sigma(\mathcal{V}_p))$
12 **else**
13 $next \leftarrow curr$
14 **end**
15 $\Delta^{v_i} \leftarrow \Delta^{v_i} \cup \{\sigma_{\mathbb{V}^\sigma \hat{\mathbb{V}}^\sigma}\}$
16 $\Gamma_{v_i} \leftarrow \Gamma_{v_i} \cup \{q_{curr} \xrightarrow{\sigma_{\mathbb{V}^\sigma \hat{\mathbb{V}}^\sigma}} q_{next}\}$
17 **end**
18 **end**
19 $H_{v_i} \leftarrow (\Delta^{v_i}, Q_{v_i}, q^\circ_{v_i}, Q^\omega_{v_i}, \Gamma_{v_i})$
20 $H' \leftarrow H' \cup \{H_{v_i}\}$
21 **end**
22 **return** H'
23 **end**

input to Algorithm 2. The result is that each event $\sigma \in \Sigma$ is replaced by the corresponding $\Delta^\sigma \in \Delta$.

3.1.3. Context switching

Note that the models resulting from Algorithm 2 require to be complemented with a mechanism to select and switch contexts. Without context switching, events of an EpFSA may occur ambiguously with respect to the source event (signal) in the DES plant, from which they have been created.

To avoid such a problem, Algorithms 3 and 4 explore the updates in $G_{\mathfrak{E}}$ to create a set, denoted H, of additional EpFSA, named *filters*. Models in H aim to implement context switching based on how variables in V change their values upon transitions.

In this paper, for the sake of clarity, the construction of H is illustrated in 2 steps: Initially, we use Algorithm 3 to calculate an intermediate set H' of filters H_{v_i}, each one addressing the context switching of a particular variable $v_i \in V$; Then, we use H', and Algorithm 4, to calculate the final set H of 2-state modular filters, such that $H_{\mathfrak{D}} = H'^{\|} = H^{\|}$.

The reasoning behind Algorithm 3 can be stated as follows. First, the set H' is initialised as empty. Then, for each variable $v_i \in V$ a EpFSA filter H_{v_i} is created and added to set H'. The process of creating each H_{v_i} is as follows: first, a set of states Q_{v_i}, representing each possible value in the domain of variable v_i, is created (line 4); second, the initial state $q^\circ_{v_i}$ is defined to include the initial value $v^o_i \in Dom(v_i)$ of the variable v_i; third, all states are marked ($Q^\omega_{v_i} = Q_{v_i}$) and sets Δ^{v_i} and Γ_{v_i} are initialised (line 6); then, all transitions and parameterised events of H_{v_i} are created based on how the variable v_i changes its values upon transitions.

In this way, each event σ of a transition $q \xrightarrow{\sigma:p} \in \Gamma$ has a corresponding parameterised event set Δ^σ. And, each parameterised event $\sigma_{\mathbb{V}^\sigma \hat{\mathbb{V}}^\sigma} \in \Delta^\sigma$ identifies: (i) which variable has changed its value upon the occurrence of σ, in $\mathbb{V}^\sigma$; and (ii) what was the value of this variable before σ, in $\hat{\mathbb{V}}^\sigma$, i.e. the current-state value. The value assumed after σ can be calculated by simply applying formula p to $\hat{\mathbb{V}}^\sigma$.

Therefore, as each state in Q_{v_i} maps a value of v_i, it is possible to create transitions in Γ_{v_i} labelled $\sigma_{\mathbb{V}^\sigma \hat{\mathbb{V}}^\sigma}$, such that: the current-state (q_{curr}) identifies the current value of v_i (i.e. $\hat{\mathbb{V}}^\sigma(v_i)$ (line 9)); and the next-state (q_{next}) identifies the value assumed by v_i after the transition (i.e. $p(\hat{\mathbb{V}}^\sigma(\mathcal{V}_p))$ (line 11)). When $q \xrightarrow{\sigma:p}$ does not changes the value of v_i, i.e. $v_i \notin \mathcal{V}'_p$, the current and next-state are assumed as the same (line 13).

The event $\sigma_{\mathbb{V}^\sigma \hat{\mathbb{V}}^\sigma} \in \Delta^\sigma$ is added to the event set Δ^{v_i} (line 15) and the created transition is added to Γ_{v_i}. The created non-modular filter model H_{v_i} is added to set H', which is returned by the end of the algorithm.

3.1.3.1. Example. Given the alphabet Δ, resulting from Algorithm 1, one can automate the design of the EpFSA filters in Figure 10, by giving the SpFSA plants in Figure 6 as input to Algorithm 3. As $V = \{x, y\}$, Algorithm 3 creates the filters H_x and H_y, for the variables x and y, respectively.

Consider the variable x, for example. A state set $Q_x = \{q_0, q_1, q_2, q_3\}$ is created to map the values of $Dom(x) = \{0, 1, 2, 3\}$ (line 4). The initial state is defined as q_0, because $x^o = 0$, and all states are marked. Then, for each transition $q \xrightarrow{\sigma:p} \in \Gamma$ in the plant model, and based on the corresponding events $\sigma_{\mathbb{V}^\sigma} \hat{\mathbb{V}}^\sigma \in \Delta^\sigma$, for $x \in \mathbb{V}^\sigma$, the transitions of H_x are created as follows:

- for $q \xrightarrow{b:p}$, with $p = x' \leftarrow x + 1$:
  - o transitions $q_0 \xrightarrow{b_{xy00}} q_1$, $q_0 \xrightarrow{b_{xy01}} q_1$, $q_0 \xrightarrow{b_{xy02}} q_1$, and $q_0 \xrightarrow{b_{xy03}} q_1$ are created, because the current-state value of x is 0 and the next-state value of x is $p(0) = 1$;
  - o transitions $q_1 \xrightarrow{b_{xy10}} q_2$, $q_1 \xrightarrow{b_{xy11}} q_2$, $q_1 \xrightarrow{b_{xy12}} q_2$, and $q_1 \xrightarrow{b_{xy13}} q_2$ are created, because the current-state value of x is 1 and the next-state value of x is $p(1) = 2$;
  - o transitions $q_2 \xrightarrow{b_{xy20}} q_3$, $q_2 \xrightarrow{b_{xy21}} q_3$, $q_2 \xrightarrow{b_{xy22}} q_3$, and $q_2 \xrightarrow{b_{xy23}} q_3$ are created, because the current-state value of x is 2 and the next-state value of x is $p(2) = 3$;

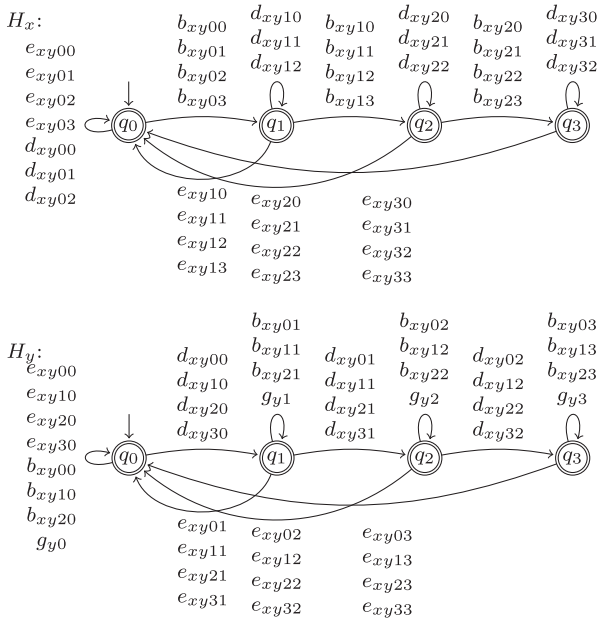


Figure 10. Non-modular filters.

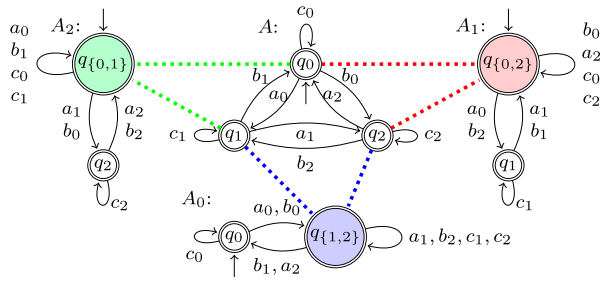


Figure 11. Example of superstates.

- for $q \xrightarrow{e:p}$, with $p = x' \leftarrow 0$:
  - o transitions $q_0 \xrightarrow{e_{xy00}} q_0$, $q_0 \xrightarrow{e_{xy01}} q_0$, $q_0 \xrightarrow{e_{xy02}} q_0$, $q_0 \xrightarrow{e_{xy03}} q_0$, $q_1 \xrightarrow{e_{xy10}} q_0$, $q_1 \xrightarrow{e_{xy11}} q_0$, $q_1 \xrightarrow{e_{xy12}} q_0$, $q_1 \xrightarrow{e_{xy13}} q_0$, $q_2 \xrightarrow{e_{xy20}} q_0$, $q_2 \xrightarrow{e_{xy21}} q_0$, $q_2 \xrightarrow{e_{xy22}} q_0$, $q_2 \xrightarrow{e_{xy23}} q_0$, $q_2 \xrightarrow{e_{xy30}} q_0$, $q_2 \xrightarrow{e_{xy31}} q_0$, $q_2 \xrightarrow{e_{xy32}} q_0$ and $q_2 \xrightarrow{e_{xy33}} q_0$ are created, because regardless the current-state value of x, the next-state value of x is always 0;
- for $\xrightarrow{d:p}$, such that p does not update x, it follows that:
  - o transitions $q_0 \xrightarrow{d_{xy00}} q_0$, $q_0 \xrightarrow{d_{xy01}} q_0$, $q_0 \xrightarrow{d_{xy02}} q_0$, $q_1 \xrightarrow{d_{xy10}} q_1$, $q_1 \xrightarrow{d_{xy11}} q_1$, $q_1 \xrightarrow{d_{xy12}} q_1$, $q_2 \xrightarrow{d_{xy20}} q_2$, $q_2 \xrightarrow{d_{xy21}} q_2$, $q_2 \xrightarrow{d_{xy22}} q_2$, $q_3 \xrightarrow{d_{xy30}} q_3$, $q_3 \xrightarrow{d_{xy31}} q_3$ and $q_3 \xrightarrow{d_{xy32}} q_3$ are created as self-loops, because regardless the current-state value of x, the next-state value is the same;

The same reasoning applies for the variable y, which leads to the EpFSA filter H_y shown in Figure 10. Then, $H_{\mathfrak{D}} = H'^{\|} = \{H_x, H_y\}$ can be exposed as set of non-modular filter models. Next we show how to obtain the modular version of H'.

3.1.3.2. Non-modular to modular conversion. The idea of *superstate* plays an essential role for the modular derivations that follow. For an FSA A, it is possible to create an FSA A', such that a group of states in A is represented by only one state in A', which is called a *superstate* (Bassino, Béal, and Perrin 1998). Each different grouping of states in A leads to a different FSA A_i, and it follows for all cases that $A \sqsubseteq A_i$, which denotes that A_i includes A and may be a more permissive model.

Figure 11 illustrates this idea by an example of an FSA A with three states (q_0, q_1 and q_2) and three 2-state models (A_0 A_1 and A_2). In A_0, the states q_1 and q_2 from A are combined into a superstate $q_{\{1,2\}}$, while state q_0 remains unchanged. Transitions between the states q_1 and q_2 in A are now self-loops on the superstate $q_{\{1,2\}}$. The same follows for A_1 and A_2, which could include two superstates $q_{\{0,2\}}$ and $q_{\{0,1\}}$. Note that $\mathcal{L}(A_0 \| A_1 \| A_2) = \mathcal{L}(A)$.

Algorithm 4: NON-MODULAR TO MODULAR FILTER CONVERSION

input : set H' of non-modular EpFSA filters
output: set H of 2-state modular EpFSA filters

1 **begin**
2 $H \leftarrow \emptyset$
3 **foreach** *model* $H'_j \in H'$, *such that* $H'_j = (\Delta'_j, Q'_j, q_j^{\circ\prime}, Q_j^{\omega\prime}, \Gamma'_j)$ **do**
4 $C \leftarrow \bigcup\{u_{ij}, c_{ij}\}$, such that $u_{ij} \neq \emptyset$, $c_{ij} \neq \emptyset$, $u_{ij} \cup c_{ij} = Q'_j$, $u_{ij} \cap c_{ij} = \emptyset$ and $|u_{ij}| = 1$
5 **foreach** $\{u_{ij}, c_{ij}\} \in C$ **do**
6 $\Delta_{ij} \leftarrow \Delta'_j$
7 $Q_{ij} \leftarrow \{q_{u_{ij}}, q_{c_{ij}}\}$
8 **if** $q_j^{\circ\prime} \in u_{ij}$ **then**
9 $q_{ij}^{\circ} \leftarrow q_{u_{ij}}$
10 **else**
11 $q_{ij}^{\circ} \leftarrow q_{c_{ij}}$
12 **end**
13 $Q_{ij}^{\omega} \leftarrow Q_{ij}$
14 $\Gamma_{ij} \leftarrow \emptyset$
15 **foreach** *transition* $q_1 \xrightarrow{\delta} q_2 \in \Gamma_{ij}$ **do**
16 **if** $q_1 \in u_{ij}$ *and* $q_2 \in u_{ij}$ **then**
17 $\Gamma_{ij} \leftarrow \Gamma_{ij} \cup \{q_{u_{ij}} \xrightarrow{\delta} q_{u_{ij}}\}$
18 **else if** $q_1 \in c_{ij}$ *and* $q_2 \in c_{ij}$ **then**
19 $\Gamma_{ij} \leftarrow \Gamma_{ij} \cup \{q_{c_{ij}} \xrightarrow{\delta} q_{c_{ij}}\}$
20 **else if** $q_1 \in u_{ij}$ *and* $q_2 \in c_{ij}$ **then**
21 $\Gamma_{ij} \leftarrow \Gamma_{ij} \cup \{q_{u_{ij}} \xrightarrow{\delta} q_{c_{ij}}\}$
22 **else**
23 $\Gamma_{ij} \leftarrow \Gamma_{ij} \cup \{q_{c_{ij}} \xrightarrow{\delta} q_{u_{ij}}\}$
24 **end**
25 **end**
26 $H_{ij} \leftarrow (\Delta_{ij}, Q_{ij}, q_{ij}^{\circ}, Q_{ij}^{\omega}, \Gamma_{ij})$
27 $H \leftarrow H \cup \{H_{ij}\}$
28 **end**
29 **end**
30 **return** H
31 **end**

The idea of superstates is now used to rebuild the components in $H'_i \in H'$. This leads to very compact models that, as shown in Section 4, have potential to benefit synthesis. The following Algorithm 4 takes as input the set H' and it constructs the set H that includes only 2-states EpFSA.

The key idea of the algorithm is established on line 4, that creates all possible combinations j of two sets of states u_{ij} and c_{ij}. We assume that u_{ij} is a unitary set including a state from Q'_i, while c_{ij} is its complement set that includes all other states from Q'_i, except u_{ij}. In conjunction, each combination j of u_{ij} and c_{ij} models a state $q_{u_{ij}}$ and a superstate $q_{c_{ij}}$ of a 2-state modular filter H_{ij}.

The reasoning behind Algorithm 4 is explained as follows. Firstly, the event set Δ_{ij} is assumed to be the same as Δ'_i. Then, the state set is created based on the two sets, u_{ij} and c_{ij}, such that $Q_{ij} = \{q_{u_{ij}}, q_{c_{ij}}\}$ (lines 6 and 7). The initial state q_{ij}° is defined as $q_{u_{ij}}$, if $q_i^{\circ\prime} \in u_{ij}$, or as $q_{c_{ij}}$ otherwise. The marked state set and the transition relation are then initialised.

For each transition $q_1 \xrightarrow{\delta} q_2 \in \Gamma_{ij}$ in H'_i, a new transition is created in H_{ij} by verifying if the states q_1 and q_2 are either the state $q_{u_{ij}}$ or combined into the superstate $q_{c_{ij}}$ (lines 17 to 23). On line 26 the modular filter H_{ij} is constructed and added to H. By repeating this procedure for all components in H', the algorithm leads to a set $H = \{H_1, \ldots, H_{ij}\}$ that can be composed to form $H_{\mathfrak{D}} = H^{\|}$. When composed to the plant model $G_{\mathfrak{D}}$, $H_{\mathfrak{D}}$ represents equivalently the context updated by variables and formulas in $G_{\mathfrak{E}}$, using a different construction.

The following proposition confirms that the proposed method preserves the same behaviour after conversion, i.e. that $G_{\mathfrak{D}} \| H_{\mathfrak{D}}$ is equivalent to the explicit version of $G_{\mathfrak{E}}$.

Proposition 3.1: *Let $G_{\mathfrak{E}}$ be a SpFSA modelling a DES plant, and let $G_{\mathfrak{D}}$ and $H_{\mathfrak{D}}$ be EpFSA computed from $G_{\mathfrak{E}}$ by Algorithms* 2, 3, *and* 4. *Then, it follows that*

$$L(G_{\mathfrak{E}}) = L(\Pi(G_{\mathfrak{D}} \| H_{\mathfrak{D}})).$$

Proof: Given a SpFSA plant model $G_{\mathfrak{E}} = (\Sigma_{\mathfrak{E}}, V_{\mathfrak{E}}, Q_{\mathfrak{E}}, q_{\mathfrak{E}}^{\circ}, Q_{\mathfrak{E}}^{\omega}, P_{V\mathfrak{E}}, \Gamma_{\mathfrak{E}})$, converted EpFSA plant $G_{\mathfrak{D}} = (\Delta_{\mathfrak{D}}, Q_{\mathfrak{D}}, q_{\mathfrak{D}}^{\circ}, Q_{\mathfrak{D}}^{\omega}, \Gamma_{\mathfrak{D}})$, and filter model $H_{\mathfrak{D}} = (\Delta_H, Q_H, q_H^{\circ}, Q_H^{\omega}, \Gamma_H)$, let $s \in \Sigma^*$ and $\sigma \in \Sigma_{\mathfrak{E}}$, such that $s\sigma \in \mathcal{L}(G_{\mathfrak{E}})$, we show that also $s\sigma \in \mathcal{L}(\Pi(G_{\mathfrak{D}} \| H_{\mathfrak{D}}))$.

First, $s\sigma \in \mathcal{L}(G_{\mathfrak{E}})$ implies, by construction, that there exists a valuation $\hat{v}_p \in Dom(\mathcal{V}_p)$ that is valid for some $p \in P_V$, which also implies that there exists $\hat{v}'_p \in Dom(\mathcal{V}'_p)$ such that, for $q_0, q_1 \in Q_{\mathfrak{E}}$, the transition $G_{\mathfrak{E}} \xrightarrow{s} (q_0, \hat{v}_p) \xrightarrow{\sigma} (q_1, \hat{v}'_p)$ exists. Also, as by assumption in this paper updates are all exact, then the valuation $\hat{v}_p$ leads to a single possible next state valuation $\hat{v}'_p$.

From Algorithm 1, the valid valuations $\hat{v}_p$ upon the event σ lead to a set $\{\delta_0, \ldots, \delta_n\} = \Delta^{\sigma}$ of events (line 11). Then, by Algorithm 2, there exists transitions $q_3 \xrightarrow{\delta_i} q_4$, for all $\delta_i \in \Delta^{\sigma}$, with $q_3, q_4 \in Q_{\mathfrak{D}}$. Furthermore, from Algorithm 4, each possible valuation is represented by a state, i.e. $\hat{v}_p$ and $\hat{v}'_p$ can be represented by states q_5 and $q_6 \in Q_H$. Because the valuations are exact, for each different valuation $\hat{v}_p$ there is only one corresponding transition $q_5 \xrightarrow{\delta_i} q_6 \in \Gamma_H$, such that $\delta_i \in \Delta^{\sigma}$. Therefore, after $\gamma \in \mathcal{L}(G_{\mathfrak{D}} \| H_{\mathfrak{D}})$, for $\Pi(\gamma) = s$, also $\gamma\delta_i \in \mathcal{L}(G_{\mathfrak{D}} \| H_{\mathfrak{D}})$, and as $\Pi(\delta_i) = \sigma$ by construction, then $s\sigma \in \mathcal{L}(\Pi(G_{\mathfrak{D}} \| H_{\mathfrak{D}}))$, which generalised shows $\mathcal{L}(G_{\mathfrak{E}}) \subseteq \mathcal{L}(\Pi(G_{\mathfrak{D}} \| H_{\mathfrak{D}}))$

For the inverse inclusion, let $\delta \in \Delta$, and $\gamma \in \Delta^*$, such that $\gamma\delta \in \mathcal{L}(G_{\mathfrak{D}} \| H_{\mathfrak{D}})$. We show that $\Pi(\gamma\delta) \in \mathcal{L}(G_{\mathfrak{E}})$. Construct $G_{\mathfrak{D}} \| H_{\mathfrak{D}} \xrightarrow{\gamma} q_0 \xrightarrow{\delta} q_1$, for $q_0, q_1 \in Q_H$. From Algorithm 4, the transition $q_0 \xrightarrow{\delta} q_1$ is constructed univocally from a set of transition (Algorithm 2) for events in Δ^{σ}, such that $\Pi(\delta_i) = \sigma$, for every $\delta_i \in \Delta^{\sigma}$

(Algorithm 1). As $\Delta^{\sigma} \neq \emptyset$, there exists at least one valid valuation so that $q_2 \xrightarrow{\sigma:p} q_3$ in $G_{\mathfrak{E}}$, for $\sigma \in \Sigma_{\mathfrak{E}}$ and $q_2, q_3 \in Q_{\mathfrak{E}}$. In the explicit notation, there exists $G_{\mathfrak{E}} \xrightarrow{\Pi(\gamma)} (q_2, \hat{\nu}_p) \xrightarrow{\sigma} (q_3, \hat{\nu}'_p)$ and $\Pi(\gamma)\sigma \in \mathcal{L}(G_{\mathfrak{E}})$. ■

3.1.3.3. Example. From the EpFSA filters in Figure 10, Algorithm 4 can calculate a set of 2-states EpFSA to be the modular filter. For instance, consider the model $H_x = (\Delta_x, Q_x, q_x^{\circ}, Q_x^{\omega}, \Gamma_x)$ and the corresponding 2-state models in Figure 12. The set of possible combinations of Q_x is $C = \{\{\{q_0\}, \{q_1, q_2, q_3\}\}, \{\{q_1\}, \{q_0, q_2, q_3\}\}, \{\{q_2\}, \{q_0, q_1, q_3\}\}, \{\{q_3\}, \{q_0, q_1, q_2\}\}\}$, that is, $u_1 = \{q_0\}$ implies $c_1 = \{q_1, q_2, q_3\}$; $u_2 = \{q_1\}$ implies $c_2 = \{q_0, q_2, q_3\}$; $u_3 = \{q_2\}$ implies $c_3 = \{q_0, q_1, q_3\}$; and $u_4 = \{q_3\}$ implies $c_4 = \{q_0, q_1, q_2\}$. Then, for each u_j there exists a 2-state filter model H_{x_j}, with construction exemplified as follows.

Given $u_1 = \{q_0\}$ and $c_1 = \{q_1, q_2, q_3\}$, the initial state is q_{u_1}, because $q_x^{\circ} \in u_1$ (line 9) and the set of states is $Q_{x_1} = Q_{x_1}^{\omega} = \{q_{u_1}, q_{c_1}\} = \{q_{\{0\}}, q_{\{1,2,3\}}\}$. Also, for each transition in Γ_x, a transition in Γ_{x_1} is created, such that:

- transitions labelled $e_{xy00}, e_{xy01}, e_{xy02}, e_{xy03}, d_{xy00}, d_{xy01}$, and d_{xy02}, are a selfloops on the state $q_0 \in Q_x$. Therefore, the created corresponding transitions on Γ_{x_1} are also a selfloops on state q_{u_1}, because $q_0 \in u_1$;
- transitions labelled $b_{xy00}, b_{xy01}, b_{xy02}$, and b_{xy03}, depart from q_0 to $q_1 \in Q_x$. Then, the created corresponding transitions in Γ_{x_1} are from q_{u_1} to q_{c_1}, because $q_0 \in u_1$ and $q_1 \in c_1$;
- transitions labelled $e_{xy10}, e_{xy11}, e_{xy12}, e_{xy13}, e_{xy20}, e_{xy21}, e_{xy22}, e_{xy23}, e_{xy30}, e_{xy31}, e_{xy32}$, and e_{xy33}, depart from q_1, q_2 or q_3, to $q_0 \in Q_x$. Then, the created corresponding transitions in Γ_{x_1} are from q_{c_1} to q_{u_1}, because $q_0 \in u_1$ and $q_1, q_2, q_3 \in c_1$;
- all other transitions from Γ_x do not leave or reach q_0. Therefore, the created corresponding transitions in Γ_{x_1} are selfloops in q_{c_1}, because $q_0 \notin c_1$;

Now, we can repeat this: for $u_2 = \{q_1\}$ and $c_2 = \{q_0, q_2, q_3\}$, which results in the filter H_{x_2}; for $u_3 = \{q_2\}$ and $c_3 = \{q_0, q_1, q_3\}$, resulting in the filter H_{x_3}; and for $u_4 = \{q_3\}$ and $c_4 = \{q_0, q_1, q_2\}$, resulting in the filter H_{x_4}. Similarly, the set of EpFSA filters $H_{y_1}, H_{y_2}, H_{y_3}, H_{y_4}$ can be obtained from H_y, and they are shown in Figure 13.

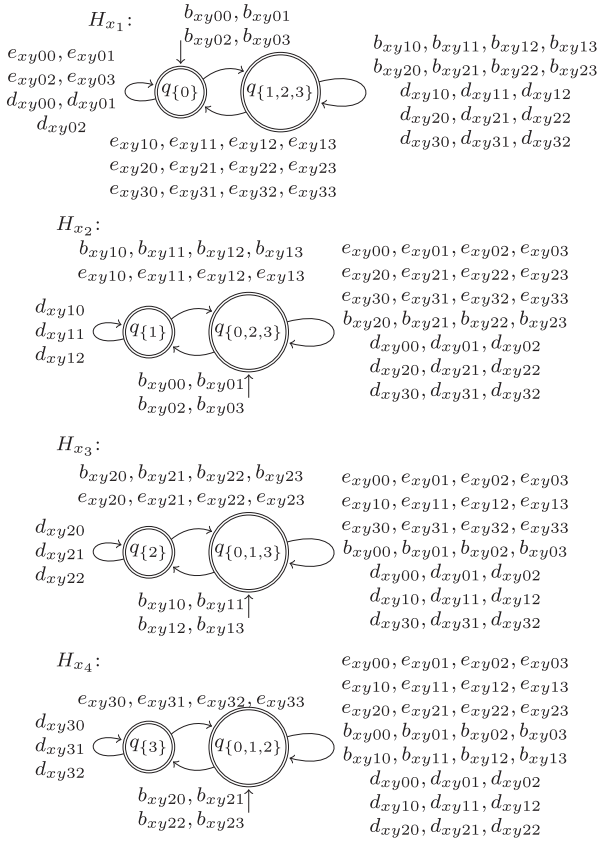


Figure 12. Modular Filters for variable *x*.

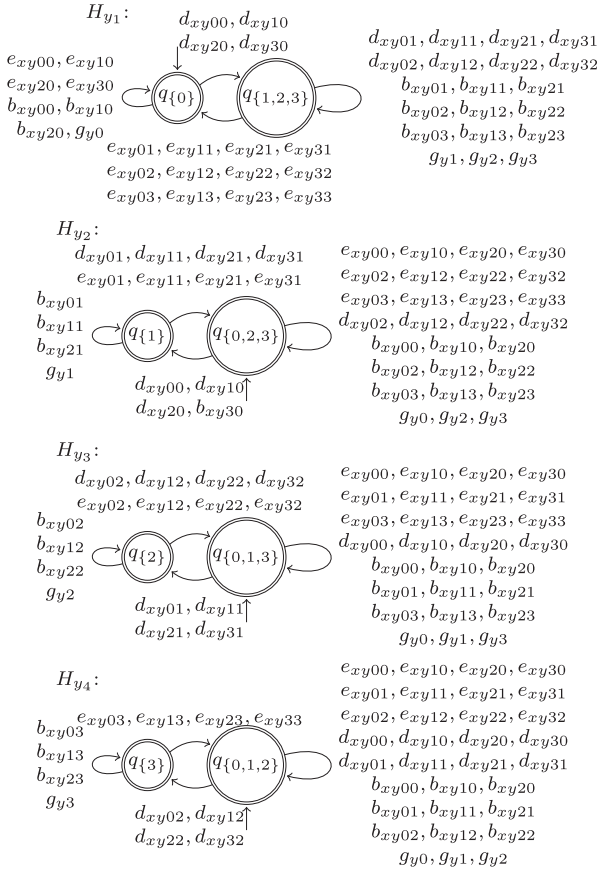


Figure 13. Modular Filters for variable *y*.

3.2. Specification conversion

One remains to show how control specifications modelled by SpFSA can be converted into EpFSA that express equivalent control rules. This construction is structured as in Algorithm 5. For each model $E_{\mathfrak{E}}{}^i$, it is created a corresponding event-parameterised model $E_{\mathfrak{D}}{}^i$ that imposes equivalent restrictions to the plant. However, instead of disabling transitions using test formulas, $E_{\mathfrak{D}}{}^i$ exploits the dilated alphabet to reproduce the same effect.

To construct $E_{\mathfrak{D}}{}^i$, the algorithm applies a very simple strategy: it reads $E_{\mathfrak{E}}{}^i$ and identifies the variable value that have been prohibited by test formulas. Then, it disables in $E_{\mathfrak{D}}{}^i$ transitions including events that correspond exactly to those variable values.

Initially, Algorithm 5 creates a structure of states identical to the input SpFSA and starts the alphabet and transition relation of $E_{\mathfrak{D}}{}^i$ as empty. Next, it reads all transitions $\xrightarrow{\sigma:p}$ from $E_{\mathfrak{E}}{}^i$ to create the alphabet and the transition relation for the output $E_{\mathfrak{D}}{}^i$. Based on each event σ read, the parameterised events in Δ^σ are added to the output alphabet (line 5). Then, transitions in the output EpFSA are created for the parameterised events $\sigma_{\mathbb{V}\hat{\mathbb{V}}}$ in Δ^σ, such that $p(\hat{\mathbb{V}}(\mathcal{V}_p)) = true$, that is, transition for which the valuation in $\hat{\mathbb{V}}$ is true for the formula p (line 7). Transitions in $E_{\mathfrak{E}}{}^i$ that do not have formulas associated to, are directly added to the set of transitions $\Gamma_{\mathfrak{D}}$ (line 9). All other are disabled, because $p(\hat{\mathbb{V}}(\mathcal{V}_p)) = false$.

3.2.1.1 Example

The SpFSA specifications of Figure 6 are given as input to Algorithm 5 and the resulting EpFSA are shown in Figure 14.

For example, $E_{\mathfrak{E}}{}^1$ enables an event e only when $x+y>0$ is true. Equivalently, $E_{\mathfrak{D}}{}^1$ disables the instance

Algorithm 5: Specification conversion from SpFSA to EpFSA

input : SpFSA specification $E_{\mathfrak{E}}{}^i = (\Sigma, V, Q, q^\circ, Q^\omega, P_V, \Gamma)$
set of parameterized events Δ
output: EpFSA specification $E_{\mathfrak{D}}{}^i = (\Delta_{\mathfrak{D}}, Q_{\mathfrak{D}}, q^\circ_{\mathfrak{D}}, Q^\omega_{\mathfrak{D}}, \Gamma_{\mathfrak{D}})$

1 **begin**
2 $Q_{\mathfrak{D}} \leftarrow Q$, $Q^\omega_{\mathfrak{D}} \leftarrow Q^\omega$, $\Delta_{\mathfrak{D}} \leftarrow \emptyset$, $\Gamma_{\mathfrak{D}} \leftarrow \emptyset$
3 $q^\circ_{\mathfrak{D}} \leftarrow q^\circ$
4 **foreach** *transition* $q_1 \xrightarrow{\sigma:p} q_2 \in \Gamma$, *such that* $q_1, q_2 \in Q$, $\sigma \in \Sigma$, $p \in P_V$ *and* $\mathcal{V}_p$ *is the set of variables in* p **do**
5 $\Delta_{\mathfrak{D}} \leftarrow \Delta_{\mathfrak{D}} \cup \Delta^\sigma$
6 **if** $\mathcal{V}_p \neq \emptyset$ **then**
7 $\Gamma_{\mathfrak{D}} \leftarrow \Gamma_{\mathfrak{D}} \cup \{q_1 \xrightarrow{\sigma_{\mathbb{V}\hat{\mathbb{V}}}} q_2\}$, for all $\sigma_{\mathbb{V}\hat{\mathbb{V}}} \in \Delta^\sigma$ such that $p(\hat{\mathbb{V}}(\mathcal{V}_p)) = true$
8 **else**
9 $\Gamma_{\mathfrak{D}} \leftarrow \Gamma_{\mathfrak{D}} \cup \{q_1 \xrightarrow{\sigma} q_2\}$, for all $\sigma \in \Delta^\sigma$
10 **end**
11 **end**
12 **return** $E_{\mathfrak{D}}{}^i$
13 **end**

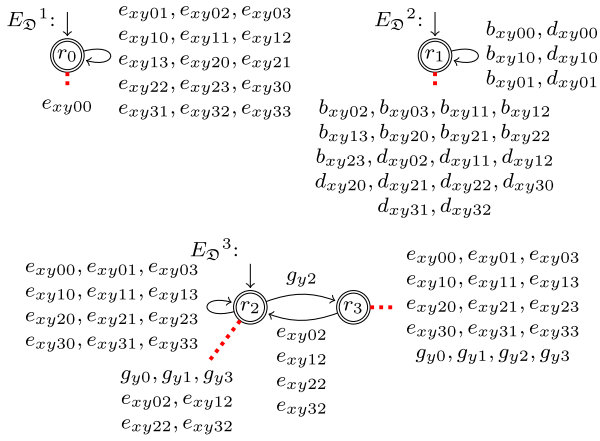


Figure 14. Converted specification models.

e_{xy00} of e, representing $\bar{x} = \bar{y} = 0$, as $p(0,0) = false$. In the same way, $E_{\mathfrak{E}}{}^2$ enables b and d when $x+y<2$ is true. Then, $E_{\mathfrak{D}}{}^2$ disables the parameterised events that model $p(\bar{x}, \bar{y}) = false$.

3.3. Test cases

Now we test our approach with respect to other examples in the literature. Four cases are selected, and they represent workflows typically found in a industrial environments, described briefly as follows:

- Example 1 (Wonham 2010): two machines interconnected by a buffer with capacity n (we fix in 2 for test). This example exploits the memory necessary to trace all possibilities of insertion and removal in a buffer;
- Example 2 (Aguiar et al. 2013): a manufacturing line where two concurrent machines produce different types of workpieces, that are then delivered to a shared 2-positions buffer. Depending on the order in which the workpieces are manufactured by the two machines, they are treated throughout the process. This means that a model for this example has to memorise every combination of workpieces all the way out the manufacturing line (Rosa, Teixeira, and Malik 2018).
- Example 3 (Zhong and Wonham 1990): an industrial *transfer line* that allows rework of material. This example remounts to Zhong and Wonham (1990), and it has been revisited several times since then by the literature. It fairly illustrates a possible blocking problem (Wonham 2010), besides to be a good measure for design and synthesis complexity (Cury et al. 2015), and modularisation alternatives (Teixeira, Cury, and de Queiroz 2018);
- Example 4 (Silva, Ribeiro, and Teixeira 2017): a flexible manufacturing system that receives two types of

Table 4. Number of states (and transitions) for literature problem models and corresponding converted models.

Domain	1	2	3	4
$G_{\mathfrak{E}}$	14 (22)	3640 (15604)	288 (888)	921600 (7937280)
$E_{\mathfrak{E}}$	1 (2)	21 (193)	4 (26)	128 (2438)
$K_{\mathfrak{E}}$	**10 (14)**	**3500 (9236)**	**624 (1180)**	**14580 (56376)**
$G_{\mathfrak{D}}$	4 (16)	40 (7104)	12 (200)	2048 (108032)
$E_{\mathfrak{D}}$	1 (6)	21 (2443)	4 (70)	128 (5632)
$H_{\mathfrak{D}}$	4 (6)	169 (546)	65 (306)	1200 (13200)
$K_{\mathfrak{D}}$	**10 (14)**	**3500 (9236)**	**624 (1180)**	**14580 (56376)**

workpieces from an external feeder. Machines and operations throughout the system are orchestrated according to the type of workpiece that has entered the line. As the system is expected to produce multiples types of workpieces in simultaneously, one has to trace every possible combination of products throughout the line, which has potential to require a huge amount of memory and modelling effort (Malik and Teixeira 2020).

The four examples were firstly modelled using SpFSA, and then converted into EpFSA using the proposed algorithms. For the sake of brevity, we do not show these models explicitly, but we summarise the results in terms of number of state and transition for the SpFSA plant ($G_{\mathfrak{E}}$), specification ($E_{\mathfrak{E}}$) and the synthesis input model $K_{\mathfrak{E}} = G_{\mathfrak{E}} \| E_{\mathfrak{E}}$; and for the respective converted EpFSA plant ($G_{\mathfrak{D}}$), specification ($E_{\mathfrak{D}}$), filter ($H_{\mathfrak{D}}$) and the synthesis input model $K_{\mathfrak{D}} = G_{\mathfrak{D}} \| E_{\mathfrak{D}} \| H_{\mathfrak{D}}$. Table 4 shows the results.

As expected, $K_{\mathfrak{E}}$ and $K_{\mathfrak{D}}$ have the same number of states and transitions for all examples, which suggests that they are equivalent, as it can be confirmed by checking, for example, language inclusion.

It remains to be shown that also synthesis results are equivalent. It is also of interest to know how easy synthesis becomes if we provide the converted EpFSA model as input, instead of SpFSA. These subjects are approached in the following.

4. Control synthesis frameworks

This Section presents synthesis frameworks that use FSA, SpFSA, and EpFSA models to obtain controllers for DESs. It also illustrates each framework by extending the example of manufacturing system in Section 4.4.

4.1. Supervisory control theory

When plant and specifications are designed as FSA G and E, a key question is to know whether G can be restricted through control in such a way that E is satisfied.

For this purpose, the event set is partitioned into the sets Σ_c and Σ_u, including *controllable* and *uncontrollable* events, respectively. Events in Σ_c can be disabled by the controller, while events in Σ_u cannot be directly prevented from occurring. Then, the *control synthesis*, a classical result of the *Supervisory Control Theory* (Ramadge and Wonham 1989), is processed to calculate a controller for G.

Given a prefix-closed plant behaviour $L(G) \subseteq \Sigma^*$ and a specification $K \subseteq L^{\omega}(G)$, it is desired to construct a so-called *supervisor* S that restricts $L(G)$ to K by disabling only controllable events. A necessary and sufficient condition for the existence of S is *controllability*: K is *controllable* with respect to $L(G)$ if $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$.

If K is controllable, then a supervisor achieving this behaviour can be implemented by an automaton V representing K, which disables any controllable event not eligible in K. If the specification is *not* controllable, then it can be reduced to the *supremal controllable sublanguage*

$$\sup\mathcal{C}(K, G) = \bigcup\{K' \subseteq K | K' \text{is controllable with respect to } L(G)\}$$

that represents the largest, or the *least restrictive*, subbehaviour of K that can be achieved by controlling G. If it is additionally nonblocking, then it is said to be an *optimal* control solution for G.

4.2. Supervisory control with SpFSA

If plant and specification of a DES are modelled as SpFSA, controllability and nonblocking have to be extended to also consider variable values, in addition to events (Teixeira et al. 2015).

For $G_{\mathfrak{E}}$ and $E_{\mathfrak{E}}$ respectively modelling plant and specification of a DES, $E_{\mathfrak{E}}$ is said to be *V-controllable* with respect to $G_{\mathfrak{E}}$ if the following holds for all $s \in \Sigma^*$, all $\mu \in \Sigma_u$, and all valuations $\hat{v}_p, \hat{v}'_p$: if $E_{\mathfrak{E}} \xrightarrow{s} (q_0, \hat{v}_v)$ and $G_{\mathfrak{E}} \xrightarrow{s} (q_1, \hat{v}_v) \xrightarrow{\mu} (q_3, \hat{v}'_v)$, then there exists $q_2 \in Q_E$ such that $E_{\mathfrak{E}} \xrightarrow{s} (q_0, \hat{v}_v) \xrightarrow{\mu} (q_2, \hat{v}'_v)$.

V -controllability differs from standard controllability in the sense that a specification must not only be able to process all uncontrollable events that are possible in the plant, but on the occurrence of an uncontrollable event it must also update the variables in the same way as the plant. Differently, for controllable events, the specification can disable some or all of the associated variable updates.

Synthesis with SpFSA is defined on the composition $E_{\mathfrak{E}} \parallel G_{\mathfrak{E}}$ and, similarly to the ordinary case, the SpFSA supervisor, denoted $\sup\mathcal{C}_V(E_{\mathfrak{E}}, G_{\mathfrak{E}})$ (Teixeira et al. 2015), represents the most permissive behaviour that can be

implemented in $G_{\mathfrak{E}}$ while satisfying $E_{\mathfrak{E}}$. Nonblocking of a SpFSA $A_{\mathrm{v}} = \left(\Sigma, V, Q, q^{\circ}, Q^{\omega}, P_V, \Gamma\right)$ can be ensured if $A_{\mathrm{v}} \xrightarrow{s} (q_0, \hat{v}_p)$ implies $(q_0, \hat{v}_p) \xrightarrow{t} (q_1, \hat{v}'_p)$ for some $q_1 \in Q^{\omega}$. For SpFSA, nonblocking is a post synthesis check, not generally included into the sup$\mathcal{C}_V$ calculation.

If, besides to be controllable, sup$\mathcal{C}_V(E_{\mathfrak{E}}, G_{\mathfrak{E}})$ is also nonblocking, then it is the optimal solution to the control problem. Under the assumption of determinism of both, events and variable values (see Teixeira et al. 2015), it can be shown that

$$\sup\mathcal{C}(\mathcal{L}(E_{\mathfrak{E}} \parallel G_{\mathfrak{E}}), \mathcal{L}(G_{\mathfrak{E}})) = \mathcal{L}(\sup\mathcal{C}_V(E_{\mathfrak{E}}, G_{\mathfrak{E}})).$$

That is, the control problem with SpFSA can be handled either using the ordinary language-based synthesis operation (sup$\mathcal{C}$) or its version extended to SpFSA (sup$\mathcal{C}_V$). In both cases, under determinism, the result is expected to be the same, so that the choice impacts only on modelling.

4.3. Supervisory control with EpFSA

When plant and specifications of a DES are given as EpFSA, synthesis algorithm is the same as for FSA. However, the resulting supervisor will control equivalently the plant only under the following assumptions.

Let the FSA plant G be given as an EpFSA $G_{\mathfrak{D}} \| H_{\mathfrak{D}}$, such that, $\Pi(G_{\mathfrak{D}} \| H_{\mathfrak{D}}) = G$ follows from (1). Let also the FSA specification E, with events in Σ, be replaced by the simpler EpFSA $E_{\mathfrak{D}}$ that uses events in Δ to express E. In this case, it is expected that, for $K = G \| E$ and $K_{\mathfrak{D}} = G_{\mathfrak{D}} \| H_{\mathfrak{D}} \| E_{\mathfrak{D}}$,

$$\mathcal{L}(\Pi(K_{\mathfrak{D}})) = \mathcal{L}(K). \tag{2}$$

From (2), and under the assumption that $H_{\mathfrak{D}}$ is *precise*, then either $K_{\mathfrak{D}}$ or K can be used as input to the classical SCT framework (Cury et al. 2015), and the resulting control solution will be such that

$$\sup\mathcal{C}(K, G) = \Pi(\sup\mathcal{C}(K_{\mathfrak{D}}, G_{\mathfrak{D}} \| H_{\mathfrak{D}})). \tag{3}$$

That is, EpFSA and FSA are equivalent under synthesis (Equation (3)), but EpFSA adds modelling benefits, at the price of ensuring Equations (1) and (2), and of designing and enforcing $H_{\mathfrak{D}}$ to be precise. As these are manual tasks, they can be complex to handle.

From our conversion method, nevertheless, we can centralise these manual steps on SpFSA formulas that are more tunned with the engineer's perception and can capture better advanced features of modern cyber-physical DES, without changing the control result. Next Section exemplifies this by using the previous examples.

Table 5. Number of states (and transitions) for literature problem models and corresponding converted models.

Domain	1	2	3	4	5
$K_{\mathfrak{E}}$	10 (14)	3500 (9236)	624 (1180)	14580 (56376)	56 (136)
$K_{\mathfrak{D}}$	10 (14)	3500 (9236)	624 (1180)	14580 (56376)	56 (136)
sup$\mathcal{C}_V$	8 (10)	742 (1816)	58 (102)	1551 (4596)	28 (57)
sup$\mathcal{C}$	8 (10)	742 (1816)	58 (102)	1551 (4596)	28 (57)

4.4. The example revisited

Now, we synthesise solutions for the four examples in Table 4, and also for the manufacturing example in Section 2.1 (labeled as 5). As shown, the solutions obtained from the converted EpFSA are the same as the ones obtained using FSA and SpFSA (Table 5).

4.5. Efficient supervisory control with EpFSA

This Section exploits additional aspects of EpFSA-based synthesis in order to reinforce the benefits of the proposed conversion mechanism.

Remark that, if on one hand the conversion method removes from the designer the burden of having to model a precise $H_{\mathfrak{D}}$, on the other hand it does not return any computational advantage. In fact, precise filters are in general associated with large state-spaces, which compensates possible simplifications in the specification model. In general, the effort to process any version of the problem, with FSA, SpFSA and EpFSA, is the same.

It has been reported (Cury et al. 2015), however, that under certain conditions synthesis with EpFSA can be simplified by including only a part of the filter $H_{\mathfrak{D}}$, while the same controller is still obtained. The plant model that abstracts part of a precise filter $H_{\mathfrak{D}}$ is called an *approximation*.

Definition 4.1: For a DES plant G, let $G_{\mathfrak{D}} = \Pi^{-1}(G)$ and let $H_{\mathfrak{D}}$ be a *precise* filter for $G_{\mathfrak{D}}$. $G_{\mathfrak{A}}$ is said to be an approximation for $G_{\mathfrak{D}} \| H_{\mathfrak{D}}$ if there exists at least one event $\sigma \in \Sigma$ in G such that $|\Gamma^{\Delta^{\sigma}}(q)| > 1$, for $q \in Q_{G_{\mathfrak{A}}}$.

Definition 4.1 implies that there exists a pair $\delta_i, \delta_j \in \Delta^{\sigma}$, such that their occurrences are not precise after at least one state in $G_{\mathfrak{A}}$. In practice this means that a context to be enabled by the plant will be ambiguous, eventually.

In control, approximations may lead to more restrictive solutions, as synthesis may not be able to distinguish contexts, so it disables any context that is unsafe. For a given approximation $G_{\mathfrak{A}}$, a precise filter $H_{\mathfrak{D}}$, and a specification $E_{\mathfrak{D}}$, such that $K_{\mathfrak{A}} = G_{\mathfrak{A}} \| E_{\mathfrak{D}}$, it follows from Cury et al. (2015) that $\sup\mathcal{C}(K_{\mathfrak{A}}, G_{\mathfrak{A}}) \cap H_{\mathfrak{D}} \subseteq \sup\mathcal{C}(K_{\mathfrak{D}}, G_{\mathfrak{D}})$.

Equality can be obtained by systematically selecting parts of $H_{\mathfrak{D}}$ to construct $G_{\mathfrak{A}}$ (Rosa, Teixeira, and

Table 6. Number of states (and transitions) for literature problem models and corresponding converted models.

Domain	1	2	3	4	5
$K_{\mathfrak{E}}$	10 (14)	3500 (9236)	624 (1180)	14580 (56376)	56 (136)
$K_{\mathfrak{D}}$	10 (14)	3500 (9236)	624 (1180)	14580 (56376)	56 (136)
$K_{\mathfrak{A}}{}'$	8 (16)	2400 (8976)	68 (502)	13680 (153616)	56 (136)
sup$\mathcal{C}_V$	8 (10)	742 (1816)	58 (102)	1551 (4596)	28 (57)
sup$\mathcal{C}$	8 (10)	742 (1816)	58 (102)	1551 (4596)	28 (57)
sup$\mathcal{C}'$	8 (10)	742 (1816)	58 (102)	1551 (4596)	28 (57)

Malik 2018). For that, it is essential that $H_{\mathfrak{D}}$ can be exposed modularly, which highlights the results in this paper. When $H_{\mathfrak{D}}$ is given modularly, as in the output of Algorithm 4, the approach in Rosa, Teixeira, and Malik (2018) selects all and only the parts of $H_{\mathfrak{D}}$ that are really necessary to synthesis. This leads to a composition $H_{\mathfrak{A}}$ of modules, for $L(H_{\mathfrak{D}}) \subseteq L(H_{\mathfrak{A}})$. Then an improved approximation $G_{\mathfrak{A}}{}' = G_{\mathfrak{D}} \| H_{\mathfrak{A}}$ can be constructed such that, for $K_{\mathfrak{A}}{}' = G_{\mathfrak{A}}{}' \| E_{\mathfrak{D}}$, it always implies $\text{sup}\mathcal{C}' = \text{sup}\mathcal{C}(K_{\mathfrak{A}}{}', G_{\mathfrak{A}}{}') \cap H_{\mathfrak{D}} = \text{sup}\mathcal{C}(K_{\mathfrak{D}}, G_{\mathfrak{D}})$.

4.5.1. Example for efficient synthesis with modular EpFSA

Finally, we synthesise a solution for each examples, with and without approximations, in order to show their equivalences and the simplification brought by synthesis.

As shown in Table 6, the solutions obtained using the input SpFSA models (row 5), using the converted EpFSA (row 6), and using approximations (row 7) are all the same. However, the cost to synthesise with approximations tends to be reduced, as it in general explores a smaller state-space. Such a reduction can be more substantial if supported by appropriate tooling to select filters to be included in synthesis which, in this paper, is suggested as a future aim.

5. Conclusion

This paper discusses how parameters can be exploited to simplify modelling and control of DESs. Two methods are presented, each one constructed on a specific formal background, so that they do not directly combine advantages. A conversion method is then proposed to allow conducting modelling using a design-friendly approach, and systematically migrate it to another framework, more tuned with synthesis and implementation issues.

The algorithms presented in the paper are all polynomials in the number of states of the input automata models. The provided tooling support allows them to be immediately used for industrial engineering tasks.

Several examples are used to illustrate the approach, some extended from the literature and others introduced in the manuscript. In particular, they share the feature of requiring the memory of intricate sequences of events in order to decide about control aspects of the DES. By using the proposed method, nevertheless, this memory becomes shorter during modelling phase, and it can be easily handled by formal structures that depend minimally on the engineer.

Prospects of future research aim to extend our approach over modular control frameworks because, large DES problems can be intractable by the monolithic versions of the SCT. We also intend to investigate more about implementation issues, besides to extend the respective tooling support.

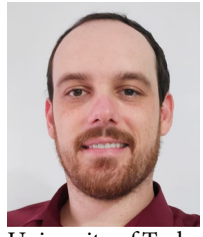
Notes

1. The SpFSA literature usually refers to *states* as *locations*, composed by a state and associated parameters, which is here generalised indistinctly.
2. The equivalence between K and $\Pi(K_{\mathfrak{D}})$ (and similarly K and $K_{\mathfrak{E}}$) follows, in general, but it depends on modelling tasks, which we assume here to be well defined from the engineering point of view.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Notes on contributors



Luiz Fernando Puttow Southier is a Ph.D. student in Applied Informatics at the Pontifical Catholic University of Paraná (PUCPR). He is a scholarship holder from the Brazilian National Council for Scientific and Technological Development (CNPq). He holds a master's degree in Electrical Engineering from the Federal University of Technology - Parana (UTFPR), having developed research in Modeling and Control of Discrete Event Systems. As a graduate in Computer Engineering, from UTFPR with a period at Pennsylvania State University, he carried out a research internship at the Department of Computer Science at the New York University, USA. In addition, he performed Scientific Initiation researching and developing Web solutions for Linear Programming and Linear Algebra, Economic Engineering and Public Management. His research interests are in the field of Process Mining, Discrete Event Systems, Economic Process Analysis and Economic Engineering, among others.

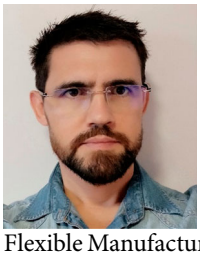
Dalcimar Casanova holds a bachelor's degree in Computer Science from the Universidade do Oeste de Santa Catarina UNOESC (2005), a master's degree in Computer Science and Computational Mathematics from the Institute of Mathematical and Computer Sciences ICMC-USP (2008), a Ph.D. candidate in Computational Physics from the Institute of Physics of São Carlos IFSC-USP (2013) and post-doctoral fellow at the Institute of

Physics of São Carlos IFSC-USP (2015). He is currently a professor at the Federal Technological University of Paraná UTFPR and permanent professor at PPGEE. He has experience in Computer Science, Computational Physics, and applications in multidisciplinary areas, working mainly on the following topics: computer vision, complex networks, fractals, machine learning and bioinformatics.

Luís Soares Barbosa is a full professor at the Computer Science Department, Universidade do Minho, and a senior researcher at the High Assurance Software laboratory, HASLab INESC TEC. Recently he has joined INL, the International Iberian Nanotechnology Laboratory, to lead the new Quantum Software Engineering Group. His main research focuses on programme semantics, logics and calculi applied to rigorous software analysis, design, and construction. Most of his work is framed on Coalgebra and Modal Logic.

César Rafael Claure Torrico holds a degree in Electrical Engineering from the Universidad Mayor de San Simón (1995), a master's degree in Electrical Engineering from the Federal University of Santa Catarina (1999) and a Ph.D. in Electrical Engineering from the Federal University of Santa Catarina (2003). He is currently a professor at the Federal Technological University of Paraná-UTFPR. He has experience in the field of Industrial Automation and Control, working mainly on the following topics: microcontrollers, discrete event systems, supervisory control, and vector control of induction motors.

Marco Barbosa graduated in Informatics at Universidade de Cruz Alta (1998). He has a master's degree in Computer Science from Universidade Federal do Rio Grande do Sul (2001) and Ph.D. at Informatics from University of Minho (2009). He has experience in Computer Science, focussing on Analysis of Algorithms and Computational Complexity, acting on the following subjects: algorithm complexity, automatic algorithm analysis, teaching support, informatics in education and coinduction.



Marcelo Teixeira is a Computer Scientist with master's in Computer Engineering and Ph.D. in Automation & Systems Engineering. His research interests cover some topics in Computer Engineering, Electrical Engineering and Automation Sciences, with special focus on Discrete-Event Systems, Cyber-Physical Systems, Flexible Manufacturing Systems, Industry 4.0, Synthesis of controllers for industrial processes, industrial automation, and automatic synthesis of software. He's been an active member of the IEEE since 2016, participating of the Industrial Electronic Society (IES), Technical Committee on Factory Automation, Subcommittee Industrial Automated Systems and Control. In 2020, he received a Research Productivity Grant from CNPq.

Data availability statement

Data sharing is not applicable to this article as no new data were created or analysed in this study.

ORCID

Marcelo Teixeira http://orcid.org/0000-0002-1008-7838

References

Aguiar, R. S. S., A. E. C. Cunha, J. E. R. Cury, and M. H. Queiroz. 2013. "Heuristic Search of Supervisors by Approximated Distinguishers." In *Dependable Control of Discrete Systems (DCDS'13)*, York, UK, 121–126.

Bassino, Frédérique, Marie-Pierre Béal, and Dominiqu Perrin. 1998. "Super-State Automata and Rational Trees. Latin American Symposium on Theoretical Informatics. Brazil.

Cassandras, Christos G, and Stephane Lafortune. 2009. *Introduction to Discrete Event Systems*. Springer, Boston, MA: Springer Science & Business Media.

Chen, Yi-Liang, and Feng Lin. 2001. "Safety control of discrete event systems using finite state machines with parameters. Proceedings of the 2001 American Control Conference. Arlington, VA, USA.

Cury, José E. R., Max Hering de Queiroz, Gustavo Bouzon, and Marcelo Teixeira. 2015. "Supervisory Control of Discrete Event Systems with Distinguishers." *Automatica* 56: 93–104.

Dotoli, Mariagrazia, Alexander Fay, Marek Miskowicz, and Carla Seatzu. 2017. "Advanced Control in Factory Automation: A Survey." *International Journal of Production Research* 55 (5): 1243–1259.

Esmaeilian, Behzad, Sara Behdad, and Ben Wang. 2016. "The Evolution and Future of Manufacturing: A Review." *Journal of Manufacturing Systems* 39: 79–100.

Gohari, P., and W. M. Wonham. 2000. "On the Complexity of Supervisory Control Design in the RW Framework." *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 30 (5): 643–652.

Guo, Daqiang, Ray Y. Zhong, Shiquan Ling, Yiming Rong, and George Q. Huang. 2020. "A Roadmap for Assembly 4.0: Self-configuration of Fixed-Position Assembly Islands under Graduation Intelligent Manufacturing System." *International Journal of Production Research* 58 (15): 4631–4646.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 1939. *Introduction to Automata Theory, Languages, and Computation*. 2nd. Boston, MA: Pearson Education.

Hu, HeSuan, and Yang Liu. 2015. "Supervisor Synthesis and Performance Improvement for Automated Manufacturing Systems by Using Petri Nets." *IEEE Transactions on Industrial Informatics* 11 (2): 450–458.

Hu, Hesuan, Yang Liu, and Ling Yuan. 2016. "Supervisor Simplification in FMSs: Comparative Studies and New Results Using Petri Nets." *IEEE Transactions on Control Systems Technology* 24 (1): 81–95.

Hu, Hesuan, Yang Liu, and Mengchu Zhou. 2015. "Maximally Permissive Distributed Control of Large Scale Automated Manufacturing Systems Modeled With Petri Nets." *IEEE Transactions on Control Systems Technology* 23 (5): 2026–2034.

Malik, Robi, and Marcelo Teixeira. 2016. "Modular Supervisor Synthesis for Extended Finite-State Machines Subject

to Controllability." In International Workshop on Discrete Event Systems (WODES), 91–96. IEEE. China.

Malik, R., and Marcelo Teixeira. 2020. "Synthesis of Least Restrictive Controllable Supervisors for Extended Finite-State Machines with Variable Abstraction." *Discrete Event Dynamic Systems* 30: 211–241.

Malik, R., and Marcelo Teixeira. 2021. "Optimal Modular Control of Discrete Event Systems with Distinguishers and Approximations." *Discrete Event Dynamic Systems* 31: 659–691.

Mohajerani, Sahar, Robi Malik, and Martin Fabian. 2017. "Compositional Synthesis of Supervisors in the Form of State Machines and State Maps." *Automatica* 76: 277–281.

Mourtzis, Dimitris. 2020. "Simulation in the Design and Operation of Manufacturing Systems: State of the Art and New Trends." *International Journal of Production Research* 58 (7): 1927–1949.

Pedrielli, Giulia, Andrea Matta, Arianna Alfieri, and Mengyi Zhang. 2018. "Design and Control of Manufacturing Systems: A Discrete Event Optimisation Methodology." *International Journal of Production Research* 56 (1–2): 543–564.

Qamsane, Yassine, Abdelouahed Tajer, and Alexandre Philippot. 2017. "A Synthesis Approach to Distributed Supervisory Control Design for Manufacturing Systems with Grafcet Implementation." *International Journal of Production Research* 55 (15): 4283–4303.

Ramadge, Peter J. G., and W. Murray Wonham. 1989. "The Control of Discrete Event Systems." *Proceedings of the IEEE* 77 (1): 81–98.

Rosa, Marcelo, Marco A. C. Barbosa, and Marcelo Teixeira. 2019. "Service-Based Manufacturing Systems: Modelling and Control." *International Journal of Production Research* 57 (11): 3421–3434.

Rosa, M., M. Teixeira, G. W. Denardin, C. R. C. Torrico, and J. E. R. Cury. 2017. "Efficient Implementation of Distinguished Controllers for Discrete-Event Systems." In *IFAC World Congress, WC'17*, Toulouse, France, 1215–1220.

Rosa, Marcelo, Marcelo Teixeira, and Robi Malik. 2018. "Exploiting Approximations in Supervisory Control with Distinguishers." In *International Workshop on Discrete Event Systems*, Sorrento, Italy.

Silva, André Lucas, Richardson Ribeiro, and Marcelo Teixeira. 2017. "Modeling and Control of Flexible Context-Dependent Manufacturing Systems." *Information Sciences* 421: 1–14.

Southier, Luiz Fernando Puttow. 2021. *Extended State Machine Converter*. Federal University of Technology Parana. https://luizsouthier.github.io/conversor.html.

Southier, Luiz F. P., Muriel Mazzetto, Dalcimar Casanova, Marco A. C. Barbosa, Luis S. Barbosa, and Marcelo Teixeira. 2019. "Combining Advantages from Parameters in Modeling and Control of Discrete Event Systems." In 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 370–377. IEEE. Spain.

Teixeira, M., J. E. R. Cury, and M. H. de Queiroz. 2018. "Exploiting Distinguishers in Local Modular Control of Discrete-Event Systems." *IEEE Transactions on Automation Science and Engineering* 15 (3): 1431–1437.

Teixeira, Marcelo, Robi Malik, José E. R. Cury, and Max H. de Queiroz. 2015. "Supervisory Control of Des with Extended Finite-State Machines and Variable Abstraction." *IEEE Transactions on Automatic Control* 60 (1): 118–129.

Wonham, W. M. 2010. *Supervisory Control of Discrete Event Systems*. Canada: University of Toronto.

Zhong, H., and W. M. Wonham. 1990. "On the Consistency of Hierarchical Supervision in Discrete-Event Systems." *IEEE Transactions on Automatic Control* 35 (10): 1125–1134.