Discrete Event Dyn Syst (2016) 26:33–84
DOI 10.1007/s10626-015-0217-y

A framework for compositional nonblocking verification of extended finite-state machines

Sahar Mohajerani[1] · Robi Malik[2] · Martin Fabian[3]

Received: 10 September 2014 / Accepted: 12 August 2015 / Published online: 12 September 2015

Abstract This paper presents a framework for *compositional nonblocking verification* of discrete event systems modelled as *extended finite-state machines* (EFSM). Previous results are improved to consider general *conflict-equivalence* based abstractions of EFSMs communicating both via shared variables and events. Performance issues resulting from the conversion of EFSM systems to finite-state machine systems are avoided by operating directly on EFSMs, deferring the unfolding of variables into state machines as long as possible. Several additional methods to abstract EFSMs and remove events are also presented. The proposed algorithm has been implemented in the discrete event systems tool Supremica, and the paper presents experimental results for several large EFSM models that can be verified faster than by previously used methods.

Keywords Extended finite-state machines · Model checking · Nonblocking · Compositional verification · Supervisory control theory

✉ Sahar Mohajerani
sahar.mohajerani@gmail.com

Robi Malik
robi@waikato.ac.nz

Martin Fabian
fabian@chalmers.se

[1] Vehicle Dynamics and Active Safety Center, Volvo Cars Corporation, Göteborg, Sweden

[2] Department of Computer Science, University of Waikato, Hamilton, New Zealand

[3] Department of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden

1 Introduction

Many discrete event systems are safety-critical, where failures can result in huge financial losses or even human fatalities. Logical correctness is a crucial property of these systems, and formal verification is an important part of guaranteeing it. This paper focuses on the verification of the *nonblocking* property (Ramadge and Wonham 1989).

Formal verification requires a formal model, and *finite-state machines (FSM)* are widely used to represent discrete event systems (Ramadge and Wonham 1989). FSMs describe the behaviour of a system using *states* and *transitions* between these states. The transitions are associated to *events* through which the FSM can interact with other FSMs and the outside world. For systems with data dependency, it is natural to extend FSMs with variables and guards. This results in *extended finite-state machines (EFSM)*, which interact through events and bounded discrete variables. EFSMs, also referred to as *extended finite-state automata (EFA)*, have been similarly defined by several researchers (Cheng and Krishnakumar 1993; Chen and Lin 2000; Sköldstam et al. 2007; Zhaoa et al. 2012; Teixeira et al. 2013).

EFSMs facilitate the modelling of complex discrete event systems that include counters or other quantitative variables. The state spaces of such systems can be huge, yet they can be modelled concisely with only a few state machine diagrams. On the other hand, the formal verification of these systems remains a challenge, because technically verification must take all possible combinations of variable values into account, often resulting in *state-space explosion*.

Various approaches have been proposed to overcome state space explosion. With *symbolic model checking*, the explicit enumeration of states is avoided using a symbolic representation (Baier and Katoen 2008; McMillan 1993), typically consisting of *ordered binary decision diagrams* (Bryant 1992), making it possible to explore much larger state spaces (Vahidi 2004).

With *compositional verification* (Graf and Steffen 1990) and *abstract interpretations* (Dams et al. 1994), the model is simplified before or during verification in an attempt to reduce combinatorial complexity. The nonblocking property considered in this paper requires special types of abstraction for compositional verification to work. Abstraction based on *conflict equivalence* (Malik et al. 2006) is more effective than general abstract interpretations (Dams et al. 1994) or bisimulation equivalence (Milner 1989). While it is well-known that nonblocking verification and similar model checking problems are NP (Gohari and Wonham 2000), and the worst-case complexity of compositional verification is even worse, experimental results (Flordal and Malik 2009; Su et al. 2010; Malik and Leduc 2013) show that compositional nonblocking verification can efficiently verify several large FSM models that cannot be verified by standard monolithic verification.

Compositional verification can also be applied to systems modelled as EFSMs, after converting the EFSMs to a set of FSMs (Sköldstam et al. 2007). However, the conversion can increase the number of events significantly, and in some cases takes longer than the verification (Mohajerani et al. 2013b). Recently, a direct method for compositional verification of EFSM models has been proposed (Mohajerani et al. 2013a; 2014), which removes the need to convert EFSMs to FSMs. In (Mohajerani et al. 2013a), *symbolic observation equivalence* is used as the only abstraction method. This is generalised to conflict equivalence in (Mohajerani et al. 2014), where a general framework for compositional verification of EFSMs communicating *only* via shared variables is introduced.

This paper is an extended version of (Mohajerani et al. 2014). It proposes a framework and an algorithm for compositional nonblocking verification of systems modelled as EFSMs that do not only communicate via shared variables but also via shared events.

- Firstly, this paper introduces *normalisation* to treat communication via shared variables (Mohajerani et al. 2014), or via events (Flordal and Malik 2009), or combinations of these, in a uniform way. After normalisation, the conflict-preserving abstractions for FSMs (Flordal and Malik 2009; Pena et al. 2009; Su et al. 2010; Malik and Leduc 2013) can be generalised for EFSM systems. While in (Mohajerani et al. 2014), *partial unfolding* of variables is limited to local variables, i.e., variables used by only one component, after normalisation this can be generalised to allow unfolding of arbitrary variables, even if they are shared between several components (Proposition 8).
- Secondly, this paper proposes more ways to simplify EFSM components while preserving the nonblocking property. In addition to the FSM-based abstraction based on (Flordal and Malik 2009; Pena et al. 2009; Su et al. 2010; Malik and Leduc 2013) (Proposition 5), four further methods (Propositions 9–10) are introduced to simplify or remove events in a normalised EFSM system, which increase the possibility of abstraction.
- Lastly, this paper combines all the abstraction methods in an algorithm (Algorithm 1) for compositional nonblocking verification of EFSM systems. The algorithm is implemented in Supremica (Åkesson et al. 2006), and has been used successfully to verify several large systems. The algorithm's performance is compared with the previously used BDD-based (Vahidi 2004) and FSM-based algorithms (Flordal and Malik 2009; Malik and Leduc 2013).

This paper is structured as follows. Section 2 introduces the notation and concepts for EFSMs, and Section 3 gives a motivating example to informally illustrate compositional nonblocking verification and abstraction of EFSM systems. Next, Section 4 explains the normalisation procedure. Then Section 5 presents the principle of compositional nonblocking verification and describes different methods to simplify EFSM systems while preserving the nonblocking property. Afterwards, Section 6 combines these results in an algorithm for compositional nonblocking verification of EFSM systems, and Section 7 presents the experimental results. Finally, Section 8 adds some concluding remarks. The Appendix contains formal proofs of all propositions contained in the paper.

2 Preliminaries

2.1 Finite-state machines

The standard means to model discrete event systems (Ramadge and Wonham 1989) are *finite-state machines (FSM)*, which synchronise on shared *events* (Hoare 1985). Events are taken from an alphabet Σ. The special *silent event* $\tau \notin \Sigma$ is not included in Σ unless explicitly mentioned using the notation $\Sigma_\tau = \Sigma \cup \{\tau\}$. Further, Σ^* is the set of all finite *traces* of events from Σ, including the *empty trace* ε. The concatenation of two traces $s, t \in \Sigma^*$ is written as st.

Definition 1 A *finite-state machine (FSM)* is a tuple $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$, where Σ

is a set of events, Q is a finite set of *states*, $\rightarrow \subseteq Q \times \Sigma_\tau \times Q$ is the *state transition relation*, $Q^\circ \subseteq Q$ is the set of *initial states*, and $Q^\omega \subseteq Q$ is the set of *marked states*.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces in Σ_τ^* by $x \xrightarrow{\epsilon} x$ for all $x \in Q$, and $x \xrightarrow{s\sigma} z$ if $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$. The transition relation is also defined for state sets $X \subseteq Q$, for example $X \xrightarrow{s} y$ means $x \xrightarrow{s} y$ for some $x \in X$.

When two or more FSMs are brought together to interact, lock-step synchronisation in the style of (Hoare 1985) is used.

Definition 2 Let $G_1 = \langle \Sigma_1, Q, \rightarrow_1, Q_1^\circ, Q_2^\omega \rangle$ and $G_2 = \langle \Sigma_2, Q_2 \rightarrow_2, Q_2^\circ, Q_2^\omega \rangle$ be two FSMs. The *synchronous composition* of G_1 and G_2 is

$$G_1 || G_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \rightarrow, Q_1^\circ \times Q_2^\circ, Q_1^\omega \times Q_2^\omega \rangle \tag{1}$$

where

$$(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2) \quad \text{if } \sigma \in \Sigma_1 \cap \Sigma_2,\ x_1 \xrightarrow{\sigma}_1 y_1, \text{ and } x_2 \xrightarrow{\sigma}_2 y_2\ ; \tag{2}$$

$$(x_1, x_2) \xrightarrow{\sigma} (y_1, x_2) \quad \text{if } \sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau\} \text{ and } x_1 \xrightarrow{\sigma}_1 y_1\ ; \tag{3}$$

$$(x_1, x_2) \xrightarrow{\sigma} (x_1, y_2) \quad \text{if } \sigma \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau\} \text{ and } x_2 \xrightarrow{\sigma}_2 y_2\ . \tag{4}$$

Hiding is the act of replacing certain events by the silent event τ.

Definition 3 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an FSM, and let $\Upsilon \subseteq \Sigma$. The result of hiding Υ from G is

$$G \setminus \Upsilon = \langle \Sigma \setminus \Upsilon, Q, \rightarrow \setminus \Upsilon, Q^\circ, Q^\omega \rangle, \tag{5}$$

where $\rightarrow \setminus \Upsilon$ is obtained from $\rightarrow$ by replacing all events in Υ with the silent event τ.

This paper concerns verification of the *nonblocking* property, which is commonly used in supervisory control theory of discrete event systems (Ramadge and Wonham 1989). A system is nonblocking if, from every reachable state, it is possible to reach a marked state, i.e., a state in Q^ω.

Definition 4 An FSM $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ is *nonblocking* if, for every trace $s \in \Sigma_\tau^*$ and every state $x \in Q$ such that $Q^\circ \xrightarrow{s} x$, there exists a trace $t \in \Sigma_\tau^*$ such that $x \xrightarrow{t} Q^\omega$.

2.2 Extended finite-state machines

Extended finite-state machines (EFSM) are similar to conventional finite-state machines, but augmented with *updates* associated to the transitions (Chen and Lin 2000; Sköldstam et al. 2007). Updates are formulas constructed from variables, integer constants, the Boolean literals *true* and *false*, and the usual arithmetic and logic connectives.

A *variable* v is an entity associated with a bounded discrete domain $\mathrm{dom}(v)$ and an initial value $v^\circ \in \mathrm{dom}(v)$. Let $V = \{v_0, \ldots, v_n\}$ be the set of variables with domain $\mathrm{dom}(V) = \mathrm{dom}(v_0) \times \cdots \times \mathrm{dom}(v_n)$. An element of $\mathrm{dom}(V)$ is also called a *valuation* and is denoted by $\hat{v} = (\hat{v}_0, \ldots, \hat{v}_n)$ with $\hat{v}_i \in \mathrm{dom}(v_i)$, and the value associated to variable $v_i \in V$ is denoted $\hat{v}[v_i] = \hat{v}_i$. The *initial valuation* is $v^\circ = (v_0^\circ, \ldots, v_n^\circ)$.

A second set of variables, called *next-state variables* and denoted by $V' = \{\, v' \mid v \in V \,\}$ with $\mathrm{dom}(V') = \mathrm{dom}(V)$, is used to describe the values of the variables after execution of a transition. Variables in V are also referred to as *current-state variables* to differentiate them from the next-state variables in V'. The set of all update formulas using variables in V and V' is denoted by Π_V.

For an update $p \in \Pi_V$, the term $\mathrm{vars}(p)$ denotes the set of all variables that occur in p, and $\mathrm{vars}'(p)$ denotes the set of all variables whose corresponding next-state variables occur in p. For example, if $p \equiv x' = y+1$ then $\mathrm{vars}(p) = \{x, y\}$ and $\mathrm{vars}'(p) = \{x\}$. Here and in the following, the relation $\equiv$ denotes syntactic identity of updates to avoid ambiguity when an update contains the equality symbol $=$. An update p without any next-state variables, $\mathrm{vars}'(p) = \emptyset$, is called a *pure guard.* Usually it is understood that variables that do not appear as next-state variables remain unchanged, and the execution of a pure guards does not change any variables. To get this interpretation, the following notion of *extension* is used.

Definition 5 Let $p \in \Pi_V$ be an update. The *extension* of p to $W \subseteq V$ is

$$\Xi_W(p) \;\equiv\; p \wedge \bigwedge\nolimits_{v \in W \setminus \mathrm{vars}'(p)} v' = v. \tag{6}$$

The extension is constructed syntactically by adding to the update p equations $v' = v$ for all variables $v \in W$ that do not already appear as next-state variables in p. For example, $\Xi_{\{x\}}(x = 1) \equiv x = 1 \wedge x' = x$ and $\Xi_{\{x,y\}}(x' = y + 1) \equiv x' = y + 1 \wedge y' = y$. Another important way to rewrite updates is *substitution*, which performs syntactic replacement of subformulas.

Definition 6 A *substitution* is a mapping $[z_1 \mapsto a_1, \ldots, z_n \mapsto a_n]$ that maps variables z_i to terms a_i. Given an update $p \in \Pi_V$, the *substitution instance* $p[z_1 \mapsto a_1, \ldots, z_n \mapsto a_n]$ is the update obtained from p by simultaneously replacing each occurrence of z_i by a_i.

For example, $(x' = x + y)[x' \mapsto 1, x \mapsto 0] \equiv 1 = 0 + y$.

With slight abuse of notation, updates $p \in \Pi_V$ are also interpreted as predicates over their variables, and they are evaluated to **F** or **T**, i.e., $p\colon \mathrm{dom}(V) \times \mathrm{dom}(V') \to \{\mathbf{F}, \mathbf{T}\}$. For example, if $V = \{x\}$ with $\mathrm{dom}(x) = \{0, 1\}$, then the update $p \equiv x' = x + 1$ means that the value of the variable x in the next state will be increased by 1 over its current-state value. Its predicate $p(x, x')$ evaluates to true as $p(0, 1) = \mathbf{T}$ and to false as $p(1, 1) = \mathbf{F}$.

Definition 7 An *extended finite-state machine (EFSM)* is a tuple $E = \langle \Sigma, Q, \to, Q^\circ, Q^\omega \rangle$, where Σ is a set of events, Q is a finite set of *locations*, $\to \subseteq Q \times \Sigma \times \Pi_V \times Q$ is the *conditional transition relation*, $Q^\circ \subseteq Q$ is the set of *initial locations*, and $Q^\omega \subseteq Q$ is the set of *marked locations*.

The expression $q_0 \overset{\sigma:p}{\to} q_1$ denotes the presence of a transition in E, from location q_0 to location q_1 with event σ and update p. Such a transition can occur if the EFSM is in location q_0 and the update p evaluates to **T**, and when it occurs, the EFSM changes its location from q_0 to q_1 while updating the variables in $\mathrm{vars}'(p)$ in accordance with p; variables not contained in $\mathrm{vars}'(p)$ remain unchanged. This can be implemented by first assigning next-state variables such that the update formula p is satisfied, and after the transition assigning the values of the next-state variables to the corresponding current-state variables.

For example, let x be a variable with domain $\mathrm{dom}(x) = \{0, \ldots, 5\}$. A transition with update $x' = x + 1$ changes the variable x by adding 1 to its current value, if it currently is less than 5. Otherwise (if $x = 5$) the transition is disabled and no updates are performed. The update $x = 3$ disables a transition unless $x = 3$ in the current state, and the value of x in the next state is not changed. Differently, the update $x' = 3$ always enables its transition, and the value of x in the next state is forced to be 3.

Given an EFSM $E = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$, its *alphabet* is also denoted by $\Sigma_E = \Sigma$. The *variable set* of E is $\mathrm{vars}(E) = \bigcup_{(q_0,\sigma,p,q_1)\in\rightarrow} \mathrm{vars}(p)$, and it contains all the variables that appear on some transitions of E.

Usually, reactive systems are modelled as several components interacting with each other. Such a model is called an *EFSM system*.

Definition 8 An *EFSM system* is a collection of interacting EFSMs,

$$\mathcal{E}E = \{E_1, \ldots, E_n\}. \tag{7}$$

The alphabet of the system $\mathcal{E}$ is $\Sigma_{\mathcal{E}} = \bigcup_{E\in\mathcal{E}} \Sigma_E$, and the variable set of $\mathcal{E}$ is $\mathrm{vars}(\mathcal{E}) = \bigcup_{E\in\mathcal{E}} \mathrm{vars}(E)$.

In the synchronisation of EFSMs, similar to FSMs, shared events between two EFSMs are synchronised in lock-step, while other events are interleaved. In addition, the updates are combined by conjunction.

Definition 9 Given two EFSMs $E_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^\circ, Q_1^\omega \rangle$ and $E_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^\circ, Q_2^\omega \rangle$, the *synchronous composition* of E_1 and E_2 is $E_1 || E_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \rightarrow_2, Q_1^\circ \times Q_2^\circ, Q_1^\omega \times Q_2^\omega \rangle$, where:

$$(x_1, x_2) \xrightarrow{\sigma:p_1\wedge p_2} (y_1, y_2) \quad \text{if } \sigma \in \Sigma_1 \cap \Sigma_2,\ x_1 \xrightarrow{\sigma:p_1}{}_1 y_1, \text{ and } x_2 \xrightarrow{\sigma:p_2}{}_2 y_2\ ; \tag{8}$$

$$(x_1, x_2) \xrightarrow{\sigma:p_1} (y_1, x_2) \quad \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ and } x_1 \xrightarrow{\sigma:p_1}{}_1 y_1\ ; \tag{9}$$

$$(x_1, x_2) \xrightarrow{\sigma:p_2} (x_1, y_2) \quad \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ and } x_2 \xrightarrow{\sigma:p_2}{}_2 y_2\ . \tag{10}$$

Using Definition 9, the global behaviour of a system $\mathcal{E} = \{E_1 \ldots E_n\}$ is expressed as $||\mathcal{E} = E_1 || \cdots || E_n$.

The standard approach to verify the nonblocking property of EFSMs evaluates all variable values. This is done by *flattening*, which introduces states for all combinations of locations and variable values (Baier and Katoen 2008).

Definition 10 Let $E = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an EFSM with variable set $\mathrm{vars}(E) = V$. The *monolithic flattened* FSM of E is $U(E) = \langle \Sigma, Q_U, \rightarrow_U, Q_U^\circ, Q_U^\omega \rangle$ where

- $Q_U = Q \times \mathrm{dom}(V)$;
- $(x, \hat{v}) \overset{\sigma}{\rightarrow}_U (y, \hat{w})$ if E contains a transition $x \overset{\sigma:p}{\rightarrow} y$ such that $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$;
- $Q_U^\circ = Q^\circ \times \{v^\circ\}$;
- $Q_U^\omega = Q^\omega \times \mathrm{dom}(V)$.

The inclusion of the variable values $\hat{v}$ in the states of the monolithic flattened FSM ensures the correct sequencing of transitions. The use of $\Xi_V(p)$ as opposed to p in the definition of $\rightarrow_U$ ensures that a variable x can only change its value if its corresponding next-state variable x' appears in the update p. The monolithic flattened FSM of an EFSM system $\mathcal{E} = \{E_1, \ldots, E_n\}$ is $U(\mathcal{E}) = U(E_1 || \cdots || E_n)$. Using these definitions, the nonblocking property is also defined for EFSMs and EFSM systems.

Definition 11 An EFSM E or an EFSM system $\mathscr{E}$ is nonblocking if the monolithic flattened FSM $U(E)$ or $U(\mathscr{E})$, respectively, is nonblocking.

3 Motivating example

This section demonstrates the process of EFSM-based compositional nonblocking verification using a simple manufacturing system modelled as interacting extended finite state-machines. The manufacturing system consists of four devices CB_1, CB_2, M_1, and M_2 as shown in Fig. 1. CB_1 and CB_2 are sections of a conveyor belt. The total capacity of the conveyor belt is given by a parameter $N \geq 1$, where it is assumed that $N = 2$ in the remainder of this section. Workpieces are loaded onto CB_1 (event l_1) from outside the system, and transported over to CB_2 (event l_2). When workpieces enter CB_2, a part detection sensor determines the type of workpieces (events p_1 and p_2). When workpieces leave CB_2, type 1 workpieces are loaded into machine M_1 (event s_1), and type 2 workpieces are loaded into machine M_2 (event s_2). The machines M_1 and M_2 then process their workpieces and output them from the system (f_1 and f_2).

The EFSM model consists of the EFSMs CB_1, CB_2, M_1, and M_2 as shown in Fig. 1. It uses variables v_1 and v_2 with domain $\{0, \ldots, N\}$ to represent the number of workpieces on conveyor section CB_1 and CB_2, respectively, and a variable t with domain $\{0, 1, 2\}$ to keep track of the type of workpiece as determined by the sensor at CB_2.

The update $v_1 + v_2 < N \land v_1' = v_1 + 1$ for event l_1 in CB_1 enforces the capacity restriction of the conveyor belt by preventing the loading of another workpiece onto CB_1 unless both conveyor sections combined have less than N workpieces, $v_1 + v_2 < N$, and if the event l_1 occurs, it increases the number of workpieces on CB_1 by 1, $v_1' = v_1 + 1$. For illustration, CB_2 contains a transition to the blocking state $\perp$ to represent that conveyor section CB_2 exceeds the capacity limit. It becomes part of the nonblocking verification to confirm that this transition is never taken.

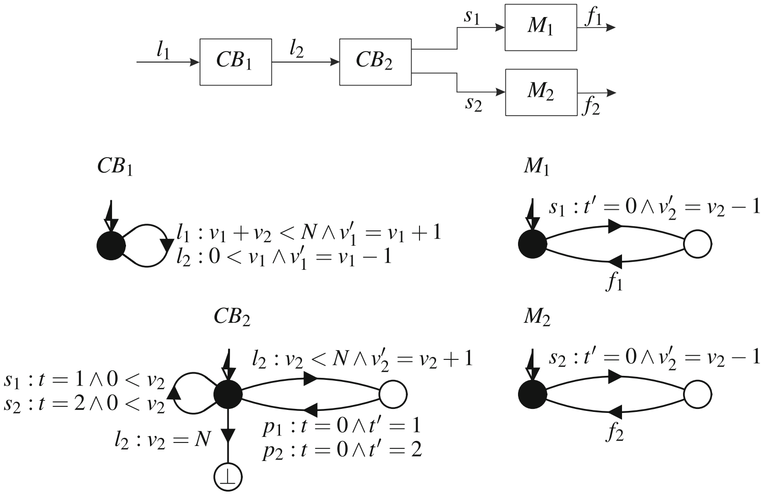

l_1
CB_1
l_2
CB_2
s_1
M_1
f_1
s_2
M_2
f_2
CB_1
$l_1 : v_1 + v_2 < N \land v_1' = v_1 + 1$
$l_2 : 0 < v_1 \land v_1' = v_1 - 1$
M_1
$s_1 : t' = 0 \land v_2' = v_2 - 1$
f_1
CB_2
$l_2 : v_2 < N \land v_2' = v_2 + 1$
$s_1 : t = 1 \land 0 < v_2$
$s_2 : t = 2 \land 0 < v_2$
$l_2 : v_2 = N$
$p_1 : t = 0 \land t' = 1$
$p_2 : t = 0 \land t' = 2$
$\perp$
M_2
$s_2 : t' = 0 \land v_2' = v_2 - 1$
f_2

Fig. 1 Manufacturing system example

The model in Fig. 1 is blocking, because the part recognition procedure is not implemented correctly in CB_2. In the following, it is demonstrated how the compositional nonblocking verification algorithm finds this fault and shows that the system is blocking without exploring the full state space.

Before EFSM-based compositional nonblocking verification starts, the preprocessing step of *normalisation* transforms the model in such a way that each event corresponds to a unique update. This facilitates reasoning about a composed system as it shows directly what effect the execution of events has on all the variables.

In order to normalise a system, the first step is to normalise individual components. The EFSM CB_2 is not normalised, because the event l_2 corresponds to two different updates $v_2 < N \land v_2' = v_2 + 1$ and $v_2 = N$. To normalise CB_2, event l_2 is replaced by two new events l_{21} and l_{22}, where the update of l_{21} is $v_2 < N \land v_2' = v_2 + 1$ and the update of l_{22} is $v_2 = N$. Having replaced l_2 in CB_2, the transition labelled with l_2 in CB_1 is replaced by two transitions labelled l_{21} and l_{22}, both of which have the update $0 < v_1 \land v_1' = v_1 - 1$ of the original l_2-transition in CB_1. These steps result in two EFSMs C_1 and C_2, shown in Fig. 2, which replace CB_1 and CB_2 in the system. This way of normalising components individually preserves the synchronous composition of the system except for the renaming of events.

Now the EFSMs are individually normalised, as each event has a unique update within each EFSM. Yet, the system as a whole is not yet normalised, because s_1 has the update $t = 1 \land 0 < v_2$ in C_2 and another update $t' = 0 \land v_2' = v_2 - 1$ in M_1. To normalise the system, the update of each event is replaced by the conjunction of the updates of the event in all the components it occurs in. For example, after normalisation the update of event s_1 becomes

$$t = 1 \land 0 < v_2 \land t' = 0 \land v_2' = v_2 - 1\ . \tag{11}$$

This conjunction is well-defined for each event since, after the first step above, events have unique updates in each component. Figure 3 shows the *normalised form* of the system. Normalisation makes it unnecessary to write the updates on the transitions. Instead, the information about the updates of the events is given in the table in Fig. 3.

Now the EFSM system is normalised, and nonblocking verification can start. This is done by constructing an abstraction of the synchronous composition of the system in several small steps. At each step, either EFSMs are abstracted and replaced by EFSMs with less transitions or locations, or variables are *unfolded*, replacing them by EFSMs and producing simpler updates. Synchronous composition is computed step by step on the abstracted EFSMs. In the end, all the variables are unfolded and the final result is a single abstracted FSM, which is simpler than the result of flattening the original EFSM system would be, while it has the same property of being nonblocking or not. Then standard monolithic nonblocking verification is applied to this abstracted FSM.

After normalisation, the system is $\mathscr{E} = \{\mathscr{N}(C_1), \mathscr{N}(C_2), \mathscr{N}(M_1), \mathscr{N}(M_2)\}$ as shown in Fig. 3. In the first step of compositional nonblocking verification, individual EFSMs are

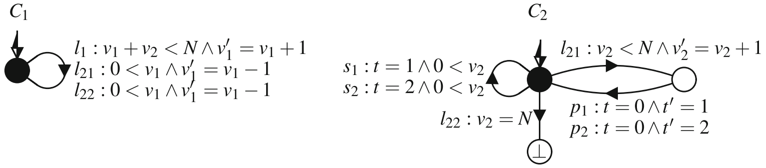
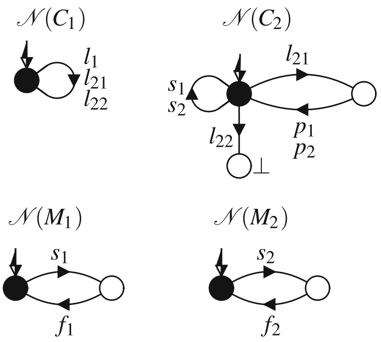

C_1
$l_1 : v_1 + v_2 < N \land v_1' = v_1 + 1$
$l_{21} : 0 < v_1 \land v_1' = v_1 - 1$
$l_{22} : 0 < v_1 \land v_1' = v_1 - 1$
C_2
$l_{21} : v_2 < N \land v_2' = v_2 + 1$
$s_1 : t = 1 \land 0 < v_2$
$s_2 : t = 2 \land 0 < v_2$
$p_1 : t = 0 \land t' = 1$
$p_2 : t = 0 \land t' = 2$
$l_{22} : v_2 = N$
$\perp$

Fig. 2 Normalised EFSMs obtained from CB_1 and CB_2



Event	Update
l_1	$v_1 + v_2 < 2 \land v_1' = v_1 + 1$
l_{21}	$0 < v_1 \land v_1' = v_1 - 1 \land v_2 < 2 \land v_2' = v_2 + 1$
l_{22}	$0 < v_1 \land v_1' = v_1 - 1 \land v_2 = 2$
s_1	$t = 1 \land 0 < v_2 \land t' = 0 \land v_2' = v_2 - 1$
f_1	*true*
s_2	$t = 2 \land 0 < v_2 \land t' = 0 \land v_2' = v_2 - 1$
f_2	*true*
p_1	$t = 0 \land t' = 1$
p_2	$t = 0 \land t' = 2$

Fig. 3 Normalised manufacturing system for $N = 2$.

abstracted if possible. Event f_1 only appears in $\mathcal{N}(M_1)$. Such events are referred to as *local events*. If the update of a local event is *true*, then transitions labelled by that event are always executable and execution will not change the value of any variable. Thus, local events corresponding to *true* updates can be *hidden*, that is, replaced by the silent event τ. After hiding the local event f_1 in $\mathcal{N}(M_1)$, the two states of $\mathcal{N}(M_1)$ can be merged using the conflict-preserving abstraction method of *observation equivalence* (Milner 1989). The same steps are applied to $\mathcal{N}(M_2)$. Fig. 4 shows the EFSMs $\mathcal{N}(M_1) \setminus \{f_1\}$ and $\mathcal{N}(M_2) \setminus \{f_2\}$ resulting from hiding and the resulting abstractions $\tilde{M}_1$ and $\tilde{M}_2$.

Events l_1, p_1, and p_2 are also local. However, these events cannot yet be hidden because of their nontrivial updates. Since the abstraction methods greatly benefit from hiding, the next step is to simplify updates of some events to make hiding possible.

The only variable in the updates of p_1 and p_2 is t. This observation suggests to unfold the variable t, removing this variable from the updates, so that the events p_1 and p_2 can be hidden and more abstraction becomes possible. Partial unfolding replaces a variable by a new EFSM, called the *variable EFSM*, which has one location for each value in the domain of the unfolded variable, and transitions that reflect the way the variable is updated by the corresponding events. The variable EFSM T for t, shown in Fig. 5, has three locations corresponding to $\mathrm{dom}(t) = \{0, 1, 2\}$. The variable t changes from 0 to 1 and from 0 to 2 by executing events p_1 and p_2, respectively, and from 1 to 0 on the occurrence of s_1, and from 2 to 0 on the occurrence of s_2. Now the updates of these events can be simplified as the variable EFSM T contains the effect the events have on the variable t. The results are shown in the table in Fig. 5: the updates of s_1 and s_2 are simplified to no longer include t, and the updates of p_1 and p_2 become *true*. However, p_1 and p_2 are no longer local events as they now appear in the variable EFSM T and in $\mathcal{N}(C_2)$.

So, now the synchronous composition $TC = T || \mathcal{N}(C_2)$ is constructed, shown in Fig. 5, which has local events p_1 and p_2 that can now be hidden because of their *true* updates. Hiding results in the EFSM $TC \setminus \{p_1, p_2\}$, also shown in Fig. 5. All the states $\perp$ can be

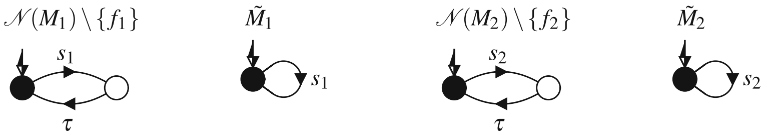
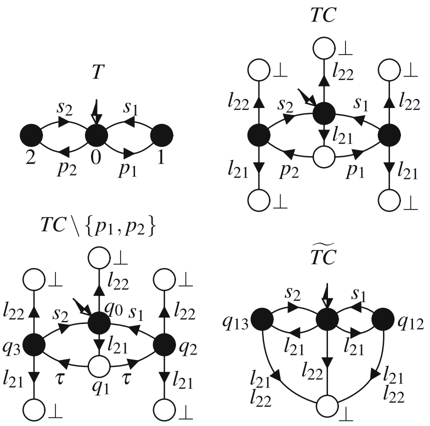

$\mathcal{N}(M_1) \setminus \{f_1\}$
s_1
τ
$\tilde{M}_1$
s_1
$\mathcal{N}(M_2) \setminus \{f_2\}$
s_2
τ
$\tilde{M}_2$
s_2

Fig. 4 Abstraction results of $\mathcal{N}(M_1)$ and $\mathcal{N}(M_2)$.

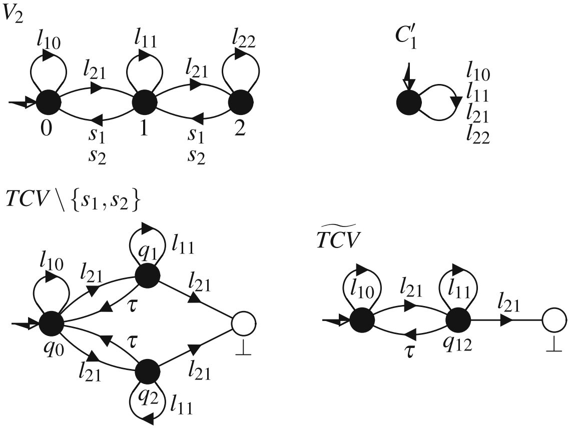


Event	Update
l_1	$v_1 + v_2 < 2 \land v_1' = v_1 + 1$
l_{21}	$0 < v_1 \land v_1' = v_1 - 1 \land v_2 < 2 \land v_2' = v_2 + 1$
l_{22}	$0 < v_1 \land v_1' = v_1 - 1 \land v_2 = 2$
s_1	$0 < v_2 \land v_2' = v_2 - 1$
s_2	$0 < v_2 \land v_2' = v_2 - 1$
p_1	*true*
p_2	*true*

Fig. 5 The components after unfolding t.

merged since the system will be blocking if it ends up in any of these states. Also, the only states that can be reached from q_1 are q_2 and q_3, and they can only be reached by the silent event τ. Such a state can be removed by the *Only Silent Outgoing Rule* as only the τ-successor states are relevant for conflict equivalence (Flordal and Malik 2009). The EFSM $TC \setminus \{p_1, p_2\}$ can thus be simplified to $\widetilde{TC}$ in Fig. 5.

Next, the composition $\widetilde{TC} \| \tilde{M}_1 \| \tilde{M}_2$ is found to be equal to $\widetilde{TC}$, and it results in the events s_1 and s_2 being local. From the table in Fig. 5, it can be observed that the updates of s_1 and s_2 only depend on the variable v_2. Thus, the variable v_2 is unfolded, which results in the variable EFSM V_2 shown in Fig. 6. The event l_1 with update $v_1 + v_2 < 2 \land v_1' = v_1 + 1$ does not change the value of the variable v_2, so it appears on two selfloop transitions in EFSM V_2. Firstly, the case $v_2 = 0$ gives rise to a selfloop on state 0 with an update that simplifies to $v_1 < 2 \land v_1' = v_1 + 1$, and secondly the case $v_2 = 1$ gives rise to a selfloop on state 1 with simplified update $v_1 < 1 \land v_1' = v_1 + 1$. To keep the system normalised after unfolding, the event l_1 is *renamed* and replaced by two new events l_{10} and l_{11} each with a unique update. This renaming also affects component $\mathscr{N}(C_1)$, where the transition



Event	Update
l_{10}	$v_1 < 2 \land v_1' = v_1 + 1$
l_{11}	$v_1 < 1 \land v_1' = v_1 + 1$
l_{21}	$0 < v_1 \land v_1' = v_1 - 1$
l_{22}	$0 < v_1 \land v_1' = v_1 - 1$
s_1	*true*
s_2	*true*

Fig. 6 The components after unfolding v_2

labelled l_1 is replaced by transitions with both the new events, resulting in C'_1 in Fig. 6. The other events l_{21}, s_1, and s_2 also have two transitions each, but their updates simplify to the same expression in each case, which means that there is no need for further renaming.

After composition of V_2 and $\widetilde{TC}||\tilde{M}_1||\tilde{M}_2 = \widetilde{TC}$, events s_1 and s_2 can be hidden, resulting in $TCV \setminus \{s_1, s_2\}$ shown in Fig. 6. Here, states q_1 and q_2 have equivalent outgoing transitions. These states can be merged using *observation equivalence* (Milner 1989), resulting in the abstraction $\widetilde{TCV}$ also shown in Fig. 6. The event l_{22} becomes always disabled after synchronisation of V_2 and TC', so it is removed from the model. This confirms that the transition $q_0 \xrightarrow{l_2} \bot$ in CB_2 never occurs.

After all these abstractions, only the EFSMs C'_1 and $\widetilde{TCV}$ and the variable v_1 remain. The final step is to unfold v_1, which results in the variable EFSM V_1 shown in Fig. 7, where all updates are *true*. The synchronous composition of C'_1, $\widetilde{TCV}$, and V_1, also shown in Fig. 7, is blocking. This essentially shows that the system blocks if a second workpiece enters CB_2 by executing l_2, before the previous workpiece is released by executing s_1 or s_2. As the final abstraction result is blocking, it is concluded that the original model in Fig. 1 is blocking. The largest component created during the compositional steps to obtain this result is TC in Fig. 5 with nine locations and ten transitions. In contrast, standard monolithic verification would have to flatten the entire system at once, which creates a blocking FSM with 44 states and 104 transitions.

This example demonstrates how EFSM-based compositional verification works. In the sequel, Section 4 explains formally the normalisation process, and Section 5 describes the abstraction methods.

4 Normalisation

The first step of the compositional nonblocking verification algorithm proposed in this paper is *normalisation*, which rewrites an EFSM system in such a way that each event has its own distinct update. This makes it possible to examine directly the effect that executing an event has on the variables, greatly simplifying the processing of EFSM systems during the later steps of compositional verification.

Definition 12 An EFSM system $\mathcal{E}$ is *normalised* if for all transitions $x_1 \xrightarrow{\sigma:p_1} x_2$ and $y_1 \xrightarrow{\sigma:p_2} y_2$ it holds that $p_1 \equiv p_2$. An EFSM E is normalised if the EFSM system $\{E\}$ is normalised.

Definition 13 For a normalised EFSM or EFSM system E, the expression $\Delta_E(\sigma)$ denotes the unique update associated with the event $\sigma \in \Sigma_E$. Moreover, for all $\sigma \in \Sigma_E$ such that there does not exist any transition $x \xrightarrow{\sigma:p} y$ in E, it is defined that $\Delta_E(\sigma) \equiv$ *false*.

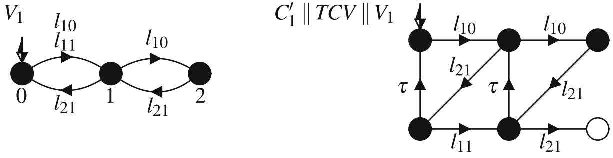


Fig. 7 The final abstracted system after unfolding v_1.

If an EFSM system $\mathcal{E}$ is normalised, then each event associates to a unique update, and $\Delta_{\mathcal{E}}(\sigma)$ is well-defined. Then the association of updates to events can be maintained separately, and EFSMs can be represented without updates on transitions, as it is done in the figures in Section 3. In a normalised EFSM system, synchronous composition also becomes simpler, because there is no need to combine update formulas.

Definition 14 Let $E_1 = \langle \Sigma_1, Q_1, \rightarrow, Q_1^{\circ}, Q_1^{\omega} \rangle$ and $E_2 = \langle \Sigma_2, Q_2, \rightarrow, Q_2^{\circ}, Q_2^{\omega} \rangle$ be EFSMs. The *normalised synchronous composition* of E_1 and E_2 is $E_1 \mathbin{\dot{\|}} E_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q2 \rightarrow, Q_1^{\circ} \times Q_2^{\circ}, Q_1^{\omega} \times Q_2^{\omega} \rangle$, where:

$$(x_1, x_2) \xrightarrow{\sigma:p} (y_1, y_2) \qquad \text{if } \sigma \in \Sigma_1 \cap \Sigma_2,\ x_1 \xrightarrow[1]{\sigma:p} y_1,\ \text{and } x_2 \xrightarrow[2]{\sigma:p} y_2\ ; \tag{12}$$

$$(x_1, x_2) \xrightarrow{\sigma:p} (y_1, x_2) \qquad \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ and } x_1 \xrightarrow[1]{\sigma:p} y_1\ ; \tag{13}$$

$$(x_1, x_2) \xrightarrow{\sigma:p} (x_1, y_2) \qquad \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ and } x_2 \xrightarrow[2]{\sigma:p} y_2\ . \tag{14}$$

In normalised synchronous composition, events and updates are treated as one entity, and synchronisation between transitions in two EFSMs is only possible when the events and updates are the same. In a normalised system, where all updates are uniquely determined by the event, this works like synchronous composition of FSMs (Definition 2): EFSMs can be composed by considering only the events, ignoring the updates. Normalised synchronous composition of a normalised EFSM system results in a normalised EFSM that produces the same flattening result as EFSM synchronous composition (Definition 9). This is confirmed by the following proposition.

Proposition 1 *Let $\mathcal{E}$ be a normalised EFSM system. Then $U(\|\mathcal{E}) = U(\dot{\|}\mathcal{E})$.*

By Proposition 1, if the system is normalised, the computation of the synchronous composition can be simplified using normalised synchronous composition. This paper concerns the verification of the nonblocking property of EFSM systems. As normalised synchronous composition produces the same flattening results as synchronous composition by Proposition 1, it follows in combination with Definition 11 that normalised synchronous composition preserves the nonblocking property of an EFSM system.

If a given EFSM system $\mathcal{E}$ is not normalised, it can be transformed into a normalised system by the two-step process of *normalisation* explained in the following. In the first step, individual EFSM components are normalised, and in the second step, the system is normalised as a whole. First, individual EFSMs are normalised by introducing new events and using a *renaming*.

Definition 15 Let Σ_1 and Σ_2 be two alphabets. A *renaming* of Σ_1 to Σ_2 is a surjective map $\rho : \Sigma_2 \rightarrow \Sigma_1$.

If an EFSM $E \in \mathcal{E}$ is not normalised, then some event σ in E is linked to more than one update. To normalise E, new events $\sigma_i \notin \Sigma_{\mathcal{E}}$ are introduced for each update p_i associated with σ and a renaming is created that maps these new events to the original event σ, i.e., $\rho(\sigma_i) = \sigma$. This results in a renamed EFSM F such that $\rho : \Sigma_F \rightarrow \Sigma_E$ and $\rho(F) = E$, and $\Delta_F(\sigma)$ is well-defined for each $\sigma \in \Sigma_F$.

Example 1 EFSM CB_2 in Fig. 1 is not normalised since l_2 corresponds to two different updates $v_2 < N \land v_2' = v_2 + 1$ and $v_2 = N$. To normalise CB_2, new events l_{21} and l_{22} are created, and the renaming ρ is introduced such that $\rho(l_{21}) = \rho(l_{22}) = l_2$ and $\rho(\sigma) = \sigma$ for all $\sigma \notin \{l_{21}, l_{22}\}$. This results in the renamed EFSM C_2 shown in Fig. 2, with $\Delta_{C_2}(l_{21}) \equiv v_2 < N \land v_2' = v_2 + 1$ and $\Delta_{C_2}(l_{22}) \equiv v_2 = N$.

After applying a renaming to some component E in a system $\mathscr{E}$, a corresponding *inverse renaming* needs to be applied to all the remaining system components $E' \neq E$ of $\mathscr{E}$, to comply with the event modification.

Definition 16 Let $E = \langle \Sigma_E, Q, \to, Q^\circ, Q^\omega \rangle$ be an EFSM, and let $\rho\colon \Sigma_E' \to \Sigma_E$ be a renaming. Then $\rho^{-1}(E) = \langle \Sigma_E', Q, \rho^{-1}(\to), Q^\circ, Q^\omega \rangle$ where $\rho^{-1}(\to) = \{\, (x, \sigma, p, y) \mid x \xrightarrow{\rho(\sigma):p} y \,\}$.

The EFSM $\rho^{-1}(E)$ is obtained by replacing transitions labelled by a replaced event σ with transitions labelled by all the events replacing it.

Example 2 After normalising CB_2 in Fig. 1, the EFSM CB_1 is replaced by $C_1 = \rho^{-1}(CB_1)$, which is obtained by replacing the l_2-transition by two transitions labelled l_{21} and l_{22}, both of which have the update $0 < v_1 \land v_1' = v_1 - 1$ of the original l_2-transition in CB_1. The EFSM $C_1 = \rho^{-1}(CB_1)$ is shown in Fig. 2.

The following proposition confirms that the structure of an EFSM system remains unchanged when a single EFSM is normalised using the combination of renaming and inverse renaming described above. The behaviour of the original system can be regained by applying the renaming to the synchronous composition of the renamed system.

Proposition 2 *Let $\mathscr{E}$ and $\mathscr{F}$ be EFSM systems, and let $\rho : \Sigma_{\mathscr{F}} \to \Sigma_{\mathscr{E}}$ be a renaming, such that $\mathscr{E} = \{E_1, E_2, \ldots, E_n\}$ and $\mathscr{F} = \{F_1\, \rho^{-1}(E_2), \ldots, \rho^{-1}(E_n)\}$ and $\rho(F_1) = E_1$. Then $\rho(||\mathscr{F}) = ||\mathscr{E}$.*

The repeated application of Proposition 2 to all components of an EFSM system with appropriate renamings results in a system where each EFSM is normalised individually. Yet, the system as a whole is not necessarily normalised as events shared between different EFSMs may be associated with different updates in their EFSMs. Therefore, a second step is needed to normalise the whole system.

Definition 17 Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ be an EFSM system such that all $E_i = \langle \Sigma_i, Q_i, \to_i, Q^\circ_\omega, Q^\omega_i \rangle$ for $1 \leq i \leq n$ are individually normalised. The *normalised form* of $\mathscr{E}$ is $\mathscr{N}(\mathscr{E}) = \{\mathscr{N}(E_1), \ldots, \mathscr{N}(E_n)\}$ where $\mathscr{N}(E_i) = \langle \Sigma_i, Q_i, \to_i^N, Q_i^\circ, Q^\omega \rangle$ and $\to_i^N = \{\, (x, \sigma, \Delta_{\mathscr{N}(\mathscr{E})}(\sigma), y) \mid x \xrightarrow[i]{\sigma:p} y \,\}$ and $\Delta_{\mathscr{N}(\mathscr{E})}(\sigma) = \bigwedge_{i:\sigma\in\Sigma_i} \Delta_{E_i}(\sigma)$.

The normalised system is obtained by assigning to each event a single update, which is the conjunction of the updates corresponding to the event in the different EFSMs. Under the assumption that the individual EFSMs in a system are individually normalised, the normalised system $\mathscr{N}(\mathscr{E})$ obtained by Definition 17 fulfils the requirement of a normalised system given in Definition 12. The update of each event after normalisation is essentially the update that would have been calculated by synchronous composition. Then, since the normalisation of individual components preserves the synchronous composition, the complete normalisation process also preserves the structure of synchronous composition of the system as a whole.

Example 3 After normalisation of the system in Section 3, the updates of the events l_{21} and l_{22} are the conjunction of the updates of these events in C_1 and C_2. For example, event l_{21} is associated with $0 < v_1 \wedge v_1' = v_1 - 1$ in C_1 according to Example 2 and with $v_2 < N \wedge v_2' = v_2 + 1$ in C_2 according to Example 1, so its update in the normalised system is $\Delta_{\mathcal{N}(\mathcal{E})}(l_{21}) \equiv 0 < v_1 \wedge v_1' = v_1 - 1 \wedge v_2 < N \wedge v_2' = v_2 + 1$ as shown in the table in Fig. 3.

The following proposition confirms that the behaviours of an EFSM system $\mathcal{E}$ and its normalised form $\mathcal{N}(\mathcal{E})$ are identical if normalised synchronous composition is used to describe the behaviour of the normalised system.

Proposition 3 *Let $\mathcal{E}$ be an EFSM system such that each $E \in \mathcal{E}$ is normalised. Then $\|\mathcal{E} = \dot{\|}\mathcal{N}(\mathcal{E})$.*

The first step to normalise a system is to normalise individual components. Proposition 2 confirms that the behaviours of a system before and after normalisation of each component are identical up to renaming of the events. As renaming preserves the nonblocking property, normalisation of individual EFSMs does not change the nonblocking property of the system. When all individual components of a system are normalised, next the operation $\mathcal{N}(\cdot)$ is applied to associate each event with a unique update in the system. Proposition 3 guarantees that the normalised system behaviour as described using normalised synchronous composition is identical up to isomorphism to the original system behaviour. Moreover, Proposition 1 and Definition 11 together guarantee that normalised synchronous composition preserves the nonblocking property. Therefore it follows from Propositions 1–3 that the normalisation procedure preserves the nonblocking property of an EFSM system. Proofs of Proposition 1–3 are given in Appendix A.

5 EFSM-based compositional verification

The objective of compositional nonblocking verification is to determine whether a normalised EFSM system

$$\mathcal{E} = \{E_1, E_2, \ldots, E_n\} \; . \tag{15}$$

is nonblocking. If a given system is not normalised, it can be normalised without affecting the nonblocking property as explained in Section 4. The straightforward approach to verify whether the system (15) is nonblocking, is to monolithically flatten the system and check for each reachable state whether it is possible to reach a marked state. However, this technique is limited by the state-space explosion problem.

In an attempt to alleviate state-space explosion, *compositional* verification (Flordal and Malik 2009) seeks to repeatedly rewrite individual components, and for example, replace E_1 in (15) by an *abstraction* F_1, and then to analyse the simpler system $\{F_1, E_2, \ldots, E_n\}$.

The abstraction steps to simplify the individual components E_i must satisfy certain conditions to guarantee that the verification result is preserved. One equivalence to support nonblocking verification is *conflict equivalence* (Malik et al. 2006).

Definition 18 Two EFSMs (or FSMs) E and F are said to be *conflict equivalent*, written $E \simeq_{conf} F$, if for any EFSM (or FSM) T, it holds that $E||T$ is nonblocking if and only if $F||T$ is nonblocking.

Due to the congruence properties of conflict equivalence (Malik et al. 2006), components of an FSM system can be replaced by conflict equivalent components while preserving the nonblocking property. This follows directly from Definition 18 when T represents the rest of the system. It is straightforward to lift this result to EFSMs.

Proposition 4 *Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ and $\mathcal{F} = \{F_1, E_2, \ldots, E_n\}$ be EFSM systems such that $E_1 \simeq_{conf} F_1$. Then $\mathcal{E}$ is nonblocking if and only if $\mathcal{F}$ is nonblocking.*

If no abstraction is possible, then some components are composed to create local events or some variables are unfolded to simplify some updates. The resulting EFSMs are abstracted again, and the procedure continues until all the variables of the systems are unfolded and the system is simple enough to be verified monolithically. To ensure correctness, all these operations are performed in such a way that the nonblocking and normalisation properties of the system are preserved.

The above-mentioned abstraction methods of FSM-based abstraction, synchronous composition, and variable unfolding are described in more detail in the following Sections 5.1, 5.2, and 5.4. Section 5.3 describes the method of update simplification, which is closely linked to variable unfolding. Finally, Section 5.5 proposes four further methods to reduce the number of events and transitions in an EFSM system.

5.1 FSM-based conflict equivalence abstraction

Compositional nonblocking verification is well-developed for systems modelled as interacting finite-state machines (Flordal and Malik 2009; Malik and Leduc 2013). Several abstraction methods for FSM-based compositional nonblocking verification with efficient implementations are known. This section shows how these methods can be applied directly to EFSMs without first flattening or unfolding any variables.

Definition 19 Let $E = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be a normalised EFSM. The *FSM form* of E is the FSM $\varphi(E) = \langle \Sigma, Q, \rightarrow_\varphi, Q^\circ, Q^\omega \rangle$, where $x \overset{\sigma}{\rightarrow}_\varphi y$ if $x \xrightarrow{\sigma:p} y$ for some $x, y \in Q$.

The FSM form of an EFSM is obtained by simply disregarding all updates. This differs from the flattened FSM (Definition 10), where variable values are expanded and updates evaluated. The conversion to FSM form and back is a straightforward operation that does not incur any blow-up, yet it makes it possible to apply FSM simplification to EFSMs. The transformation relies on the system being normalised, because in a normalised system each event has a unique update, making it possible to disregard the updates.

Most abstraction methods defined for FSMs (Flordal and Malik 2009) use *local* events, i.e., events that only appear in one FSM in the system. Local events can be hidden because they do not interact with other components of the system. In an EFSM system, even though local events are not shared with other components, the variables in their updates can still result in interaction; only local events with *true* update can be hidden. Therefore, when transforming an EFSM to an FSM, the local events with *true* updates are collected in a set Υ and hidden from the transformed FSM. Then the FSM is simplified based on the following result.

Proposition 5 *Let $\mathscr{E} = \{E_1, E_2, \ldots, E_n\}$ be a normalised EFSM system and let $\Upsilon \subseteq \Sigma_1$ such that $(\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset$ and $\Delta_{\mathscr{E}}(\sigma) \equiv$ true for all $\sigma \in \Upsilon$. Let $\mathscr{F} = \{F_1, E_2, \ldots, E_n\}$ be a normalised EFSM system such that $\varphi(E_1) \setminus \Upsilon \simeq_{conf} \varphi(F_1) \setminus \Upsilon$. Then $\mathscr{E}$ is nonblocking if and only if $\mathscr{F}$ is nonblocking.*

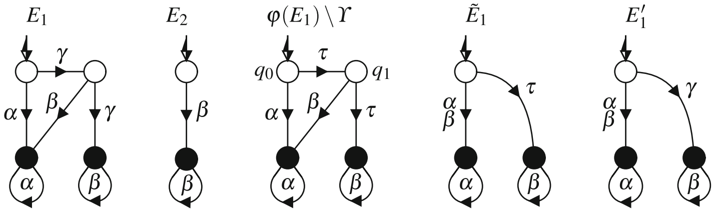
To summarise Proposition 5, to apply the conflict-preserving abstraction defined for FSM on EFSM, first the set Υ is identified as the set of events with *true* updates that appear only in the EFSM E_1 to be simplified. Next, the FSM form of E_1 is obtained by disregarding all updates, and the events in Υ are hidden from the FSM form $\varphi(E_1)$. Then the conflict equivalence abstraction methods developed for FSMs are used to simplify $\varphi(E_1)$, resulting in an FSM $\varphi(F_1)$. This FSM is interpreted as an EFSM, F_1, which is added back to the system. Proposition 5 provides a general method to simplify an EFSM in a normalised system while preserving the nonblocking property. A proof is given in Appendix B.1.

Example 4 Consider the normalised EFSM system $\mathscr{E} = \{E_1, E_2\}$ in Fig. 8. It can be observed from Fig. 8 that the events α and γ only appear in the EFSM E_1, and $\Delta_{\mathscr{E}}(\gamma) \equiv$ *true*. Therefore, the set of local events to be hidden is chosen as $\Upsilon = \{\gamma\}$. To apply the abstraction methods defined for FSMs, the EFSM E_1 is transformed to the FSM form $\varphi(E_1)$ based on Definition 19, and event γ is hidden, resulting in $\varphi(E_1) \setminus \Upsilon$ shown in Fig. 8. The two states q_0 and q_1 in $\varphi(E_1) \setminus \Upsilon$ can now be merged using the conflict-preserving *Silent Continuation Rule* (Flordal and Malik 2009), resulting in the abstracted FSM $\tilde{E}_1$ in Fig. 8. Afterwards, the FSM $\tilde{E}_1$ is transformed back to an EFSM with the same structure as $\tilde{E}_1$ and the silent event τ is replaced by the local event γ as shown in E'_1.

To ensure normalisation, the silent event τ is only used in FSMs and not in EFSMs. Therefore, the τ events are replaced by ordinary events when the FSM is converted back to an EFSM. Any local event with *true* update can be used for this replacement, and an original EFSM whose FSM form contains a τ-transition must contain at least one such event.

5.2 Partial composition

Synchronous composition is the simplest step in compositional nonblocking verification. It is always possible to replace some components of an EFSM system by their composition. This operation does not reduce the state space, but it is necessary when all other means of simplification have been exhausted. The following result follows directly from the definitions. The EFSM systems before and after normalised synchronous composition are not only equivalent with respect to nonblocking, but also identical up to isomorphism.



Event	Update
α	$y' = y + 1$
β	$y = 1$
γ	*true*

Fig. 8 Example of FSM-based conflict equivalence abstraction.

Proposition 6 *(Partial Composition)* *Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ be an EFSM system, and let $\mathscr{F} = \{E_1 \mathbin{\dot{\|}} E_2, E_3, \ldots, E_n\}$. Then $\|\mathscr{E} = \dot{\|}\mathscr{F}$.*

Proposition 6 states that the composition of two EFSMs in an EFSM system does not change the behaviour of the system. It is also clear that the system stays normalised as normalised synchronous composition does not affect the updates of events. In combination with Proposition 1, it is guaranteed that the nonblocking property of the system is preserved after composing a part of a system using this operation. A proof is given in Appendix B.2.

5.3 Update simplification

An important aspect to reasoning about EFSM systems is the symbolic manipulation of updates, in order to keep the formulas as simple as possible. This is achieved by rewriting updates into logically equivalent updates, while keeping in mind that variables that do not appear as next-state variables in an update remain unchanged in a normalised system.

Definition 20 Let $p, q \in \Pi_V$ be two updates.

- p and q are *logically equivalent*, written $p \Leftrightarrow q$, if it holds that $p(\hat{v}, \hat{w}) = q(\hat{v}, \hat{w})$ for all valuations $\hat{v}, \hat{w} \in \mathrm{dom}(V)$.
- p and q are *logically equivalent with respect to $W \subseteq V$*, written $p \Leftrightarrow_W q$, if $\Xi_W(p)$ and $\Xi_W(q)$ are logically equivalent.

Example 5 The updates $p \equiv x = 1$ and $q \equiv x = 1 \wedge x' = 1$ are not logically equivalent, because for valuations $\hat{v}[x] = 1$ and $\hat{w}[x] = 0$, e.g., it holds that $p(\hat{v}, \hat{w}) = \mathbf{T}$ but $q(\hat{v}, \hat{w}) = \mathbf{F}$. Yet, bearing in mind that the update $p \equiv x = 1$ leaves the variable $x \notin \mathrm{vars}'(p)$ unchanged, these two updates have the same effect. The extension $\Xi_{\{x\}}(p) \equiv x = 1 \wedge x' = x$ is logically equivalent to $q \equiv x = 1 \wedge x' = 1$, and therefore p and q are logically equivalent with respect to $W = \{x\}$, i.e, $p \Leftrightarrow_{\{x\}} q$.

Two updates being logically equivalent does not necessarily mean that they are logically equivalent with respect to W, nor does the converse hold in general.

Updates can be simplified automatically by standard methods of propositional reasoning, theorem proving, or binary decision diagrams (Huth and Ryan 2004; Bryant 1992). The following proposition confirms that replacing updates by updates logically equivalent with respect to the full variable set V does not effect the nonblocking property of the system.

Proposition 7 (Update Simplification) *Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ and $\mathscr{F} = \{F_1, \ldots, F_n\}$ be normalised EFSM systems with $E_i = \langle \Sigma_i, Q_i, \rightarrow_i^E, Q_i^\circ, Q^\omega[i] \rangle$ and $F_i = \langle \Sigma_i, Q_i, \rightarrow_i^F, Q_i^\circ, Q_i^\omega \rangle$. Let $V = vars(\mathscr{E}) = vars(\mathscr{F})$ and $\Delta_{\mathscr{E}}(\sigma) \Leftrightarrow_V \Delta_{\mathscr{F}}(\sigma)$ for all $\sigma \in \Sigma_{\mathscr{E}} = \Sigma_{\mathscr{F}}$, and $\rightarrow_i^F = \{\, (x, \sigma, \Delta_{\mathscr{F}}(\sigma), y) \mid x \xrightarrow{\sigma:\Delta_{\mathscr{E}(\sigma)}}{}_i^E\; y \,\}$. Then $\mathscr{E}$ is nonblocking if and only if $\mathscr{F}$ is nonblocking.*

The EFSMs in $\mathscr{E}$ and $\mathscr{F}$ in Proposition 7 have the same events and states, the only difference is that updates in $\mathscr{E}$ are replaced in $\mathscr{F}$ by updates logically equivalent with respect to all variables, maintaining normalisation by consistently making the same replacement for each event. A proof of Proposition 7 is given in Appendix B.3.

5.4 Variable unfolding

This section describes the abstraction that removes a variable from an EFSM system. It has been shown (Mohajerani et al. 2014) that unfolding a variable that only appears in one EFSM, called a *local variable*, preserves conflict equivalence. Here, this idea of partial unfolding is generalised to allow unfolding an arbitrary variable, even if it is shared between several EFSMs. In a normalised EFSM system, this can be achieved by replacing the variable with a *variable EFSM* that keeps track of the variable values.

Definition 21 Let z be a variable, and let Σ be a set of events. The *variable alphabet* of z with respect to Σ is $U_z(\Sigma) = \Sigma \times \mathrm{dom}(z) \times \mathrm{dom}(z)$.

Definition 22 Let $\mathscr{E}$ be a normalised EFSM system. The *normalised variable EFSM* of $z \in \mathrm{vars}(\mathscr{E})$ is

$$U_{\mathscr{E}}(z) = \langle U_z(\Sigma_z), \mathrm{dom}(z), \rightarrow_z, \{z_\circ\}, \mathrm{dom}(z)\rangle \tag{16}$$

where

$$\Sigma_z = \{\, \sigma \in \Sigma_{\mathscr{E}} \mid z \in \mathrm{vars}(\Delta_{\mathscr{E}}(\sigma)) \,\}\,; \tag{17}$$

$$\begin{aligned}\rightarrow_z = \{&(a, (\sigma, a, b), \Xi_{\{z\}}(\Delta_{\mathscr{E}}(\sigma))[z \mapsto a, z' \mapsto b], b) \mid \\ &z \in \mathrm{vars}(\Delta_{\mathscr{E}}(\sigma)) \text{ and } a, b \in \mathrm{dom}(z)\}.\end{aligned} \tag{18}$$

Furthermore, the *variable renaming map* $\rho_z \colon \Sigma_{\mathscr{E}} \cup U_z(\Sigma_z) \rightarrow \Sigma_{\mathscr{E}}$ for z is defined by $\rho_z(\sigma) = \sigma$ for all $\sigma \in \Sigma_{\mathscr{E}}$ and $\rho_z((\sigma, a, b)) = \sigma$ for all $(\sigma, a, b) \in U_z(\Sigma_z)$.

The variable EFSM uses the domain of the unfolded variable z as its set of locations. Events σ that have the variable z in their update are modified to have the form of (σ, a, b) to keep track of the value of the variable when the event is executed, where $a, b \in \mathrm{dom}(z)$ are the values of z before and after the transition, respectively. If z' does not appear in the update $\Delta_{\mathscr{E}}(\sigma)$, then the extension operation $\Xi_{\{z\}}$ adds the condition $z' = z$ to ensure that the value of z remains unchanged. Afterwards, the substitution $[z \mapsto a, z' \mapsto b]$ inserts the variable values corresponding to the source and target states of the transitions into the updates. The resulting updates can typically be simplified using Proposition 7.

For example, unfolding the variable v_2 with domain $\{0, 1, 2\}$ in the example of Section 3 results in the variable EFSM V_2 with three locations 0, 1, and 2 as shown in Fig. 6. By executing l_1 with update $v_1 + v_2 < 2 \wedge v_1' = v_1 + 1$, the value of v_2 does not change. Event l_1 is replaced by two new events $(l_1, 0, 0)$ and $(l_1, 1, 1)$, which are presented as l_{10} and l_{11} in Section 3. The update of $(l_1, 0, 0)$ is $(v_1 + v_2 < 2 \wedge v_1' = v_1 + 1)[v_2 \mapsto 0, v_2' \mapsto 0] \equiv (v_1 + 0 < 2 \wedge v_1' = v_1 + 1)$, which can be simplified to $v_1 < 2 \wedge v_1' = v_1 + 1$ using Proposition 7.

When a variable is unfolded, updates are changed and new events are introduced. Then the following event replacement operation $U_z(E)$ is used to ensure that the EFSMs in the system after partial unfolding use the new events. This operation replaces transitions labelled with the original events σ by new transitions labelled with each of the new events (σ, a, b), in a similar way as the inverse renaming ρ^{-1} in Definition 16.

Definition 23 Let $E = \langle \Sigma_E, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an EFSM, and let z be a variable. The *expansion* of E after unfolding z is defined by

$$\begin{aligned} U_z(E) &= \langle \Sigma^U, Q, \rightarrow^U, Q^\circ, Q^\omega \rangle \, ; && (19) \\ \Sigma^U &= U_z(\Sigma_E \cap \Sigma_z) \cup (\Sigma_E \setminus \Sigma_z) && (20) \\ \rightarrow^U &= \big\{ (x, (\sigma, a, b), \Xi_{\{z\}}(\Delta_{\mathscr{E}}(\sigma))[z \mapsto a, z' \mapsto b], y) \mid \\ & \quad z \in \text{vars}(\Delta_{\mathscr{E}}(\sigma)) \text{and} x \xrightarrow{\sigma:\Delta_{\mathscr{E}}(\sigma)} y \text{and} \sigma \in \Sigma_E \cap \Sigma_z \big\} \cup \\ & \quad \{ (x, \sigma, p, y) \mid x \xrightarrow{\sigma:p} y \text{ and } \sigma \in \Sigma_E \setminus \Sigma_z \} \, . && (21) \end{aligned}$$

Example 6 Figure 6 in Section 3 shows the expansion $C_1' = U_{v_2}(\mathscr{N}(C_1))$, which replaces the EFSM $\mathscr{N}(C_1)$ in Fig. 3 after unfolding v_2.

Given these definitions, a variable z is unfolded by adding the variable EFSM $U_{\mathscr{E}}(z)$ to the system, and replacing all EFSMs E by their expansions $U_z(E)$.

Definition 24 Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system and $z \in \text{vars}(\mathscr{E})$. The result of unfolding z in $\mathscr{E}$ is

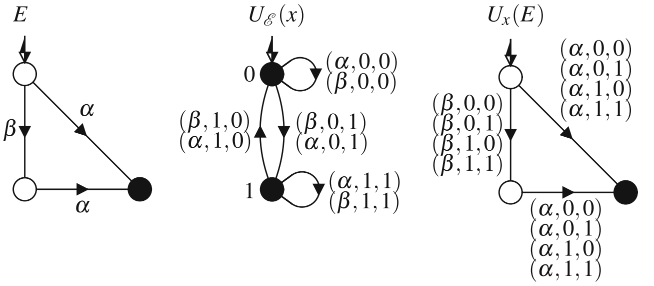
$$\mathscr{E} \setminus z = \{U_{\mathscr{E}}(z), U_z(E_1), \ldots, U_z(E_n)\} \qquad (22)$$

where $U_{\mathscr{E}}(z)$ is defined as in Definition 22 and $U_z(E_i)$ is defined as in Definition 23.

Proposition 8 ***(Variable Unfolding)*** *Let $\mathscr{E}$ be a normalised EFSM system, and let $z \in$ vars($\mathscr{E}$). Then $\mathscr{E}$ is nonblocking if and only if $\mathscr{E} \setminus z$ is nonblocking.*

Proposition 8 confirms that the nonblocking property of the system is preserved after unfolding a variable. A proof can be found in Appendix B.4.

Example 7 Consider the EFSM E in Fig. 9 with updates shown in the table, which is part of a normalised system $\mathscr{E}$. Assume dom(x) = dom(y) = {0, 1} and $x^\circ = y^\circ = 0$. Unfolding the variable x results in the variable EFSM $U_{\mathscr{E}}(x)$ with locations 0 and 1 in Fig. 9.



Event	Update
α	$y' = x + 1$
β	$x' = y + 1$
$(\alpha,0,0)$	$y' = 1$
$(\alpha,0,1)$	*false*
$(\alpha,1,0)$	*false*
$(\alpha,1,1)$	*false*
$(\beta,0,0)$	*false*
$(\beta,0,1)$	$y = 0$
$(\beta,1,0)$	*false*
$(\beta,1,1)$	$y = 0$

Fig. 9 Example of unfolding a variable x

The event α with update $y' = x + 1$ is replaced by four new events:

$$\begin{array}{lll}
(\alpha, 0, 0) & \text{with update} & \Xi_{\{x\}}\ (y'{=}x{+}1)[x \mapsto 0, x' \mapsto 0] \equiv y'{=}0{+}1 \land 0{=}0 \Leftrightarrow_V y'{=}1 \\
(\alpha, 0, 1) & \text{with update} & \Xi_{\{x\}}\ (y'{=}x{+}1)[x \mapsto 0, x' \mapsto 1] \equiv y'{=}0{+}1 \land 1{=}0 \Leftrightarrow_V \mathit{false} \\
(\alpha, 1, 0) & \text{with update} & \Xi_{\{x\}}\ (y'{=}x{+}1)[x \mapsto 1, x' \mapsto 0] \equiv y'{=}1{+}1 \land 0{=}1 \Leftrightarrow_V \mathit{false} \\
(\alpha, 1, 1) & \text{with update} & \Xi_{\{x\}}\ (y'{=}x{+}1)[x \mapsto 1, x' \mapsto 1] \equiv y'{=}1{+}1 \land 1{=}1 \Leftrightarrow_V \mathit{false}
\end{array}$$

The update of the event $(\alpha, 1, 1)$ is *false* because $y' = 2$ is not possible as $\mathrm{dom}(y) = \{0, 1\}$. Similarly, event β is replaced by four new events. The new events and their simplified updates are shown in the table in Fig. 9. Since partial unfolding introduces new events to the system, the EFSM E needs to be modified to use the new events. Thus, E is replaced by $U_x(E)$, also shown Fig. 9.

5.5 Event Simplification

Generally, reducing the number of events in a system increases the possibility of abstraction. This section proposes four different methods to reduce the number of events.

It can happen after abstraction and simplification that the updates of some events become *false*. Such events can be removed from the system and their transitions deleted. This event removal is implemented by the following operation of *restriction*.

Definition 25 The *restriction* of EFSM $E = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ to $\Omega \subseteq \Sigma$ is $E_{|\Omega} = \langle \Omega, Q, \rightarrow_{|\Omega}, Q^\circ, Q^\omega \rangle$ where

$$\rightarrow_{|\Omega} = \{\, (x, \sigma, p, y) \in \rightarrow \mid \sigma \in \Omega \,\} \,. \tag{23}$$

The restriction of $\mathcal{E} = \{E_1, \ldots, E_n\}$ to Ω is $\mathcal{E}_{|\Omega} = \{E_{1|\Omega}, \ldots, E_{n|\Omega}\}$.

Proposition 9 (false-Removal) *Let $\mathcal{E}$ be a normalised EFSM system, and let $\Lambda \subseteq \Sigma_{\mathcal{E}}$ be a set of events such that for all $\lambda \in \Lambda$ at least one of the following conditions holds:*

(i) $\Delta_{\mathcal{E}}(\lambda) \equiv$ *false;*

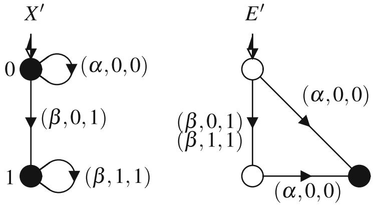
(ii) *There exists $E \in \mathcal{E}$ such that $\lambda \in \Sigma_E$, but there does not exist any transition $x \xrightarrow{\lambda:p} y$ in E.*

Then $\mathcal{E}$ is nonblocking if and only if $\mathcal{E}_{|\Sigma_{\mathcal{E}} \setminus \Lambda}$ is nonblocking.

Based on Proposition 9, events with *false* updates can be removed from the system while preserving the nonblocking property. Likewise, events that in some EFSM do not appear on any transition and therefore are always disabled, can be removed. A proof is given in Appendix B.5.

Example 8 Consider again the EFSMs $U_x(E)$ and $U_{\mathcal{E}}(x)$ in Fig. 9. The updates of events $(\alpha, 0, 1)$, $(\alpha, 1, 0)$, $(\alpha, 1, 1)$, $(\beta, 0, 0)$, and $(\beta, 1, 0)$ are *false*. Thus, these events an their transitions can be entirely removed from the system. Fig. 10 shows the simplified EFSMs X' and E' obtained from $U_{\mathcal{E}}(x)$ and $U_x(E)$, respectively, by removing the events with *false* updates.

In compositional nonblocking verification of FSM systems, events that only appear on selfloops are immediately removed because no state change is possible by these transitions. This is not possible in an EFSM system. Even though no location change is possible by



Event	Update
$(\alpha,0,0)$	$y' = 1$
$(\beta,0,1)$	$y = 0$
$(\beta,1,1)$	$y = 0$

Fig. 10 Example of removing events with *false* update.

selfloop-only events, if the updates contain next-state variables, the execution of these events can still change the system state by changing the variable values. An event can be removed from an EFSM system if it causes no location changes, which means it appears only on selfloop transitions, *and* if it causes no changes of variable values, which is guaranteed if the update of the event is a pure guard, i.e., contains no primed variables.

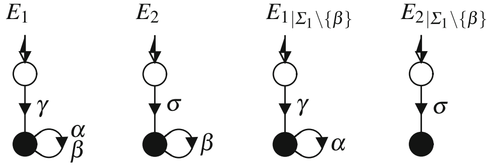
Definition 26 An EFSM $E = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ is *selfloop-only* for $\lambda \in \Sigma$ if $x \xrightarrow{\lambda:p} y$ implies $x = y$ and $\mathrm{vars}'_{\mathscr{E}}(p) = \emptyset$. EFSM E is selfloop-only for $\Lambda \subseteq \Sigma$ if E is selfloop-only for each $\lambda \in \Lambda$. An EFSM system $\mathscr{E}$ is selfloop-only for $\Lambda \subseteq \Sigma$ if each $E \in \mathscr{E}$ is selfloop-only for Λ.

An EFSM is selfloop-only for an event, if the event appears only in selfloops and the update of the event is a pure guard. The following proposition confirms that selfloop-only events can be removed without affecting the nonblocking property of an EFSM system. A proof can be found in Appendix B.5.

Proposition 10 (Selfloop Removal) *Let $\mathscr{E}$ be a normalised EFSM system that is selfloop-only for $\Lambda \subseteq \Sigma_{\mathscr{E}}$. Then $\mathscr{E}$ is nonblocking if and only if $\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda}$ is nonblocking.*

Example 9 Consider the normalised EFSM system $\mathscr{E} = \{E_1, E_2\}$ in Fig. 11, where the alphabets of E_1 and E_2 are $\Sigma_1 = \{\alpha, \beta, \gamma\}$ and $\Sigma_2 = \{\beta, \sigma\}$. Assume $\mathrm{dom}(x) = \mathrm{dom}(y) = \{0, 1\}$. The events α and β only appear on selfloops in the entire system, and the update of β is pure guard. Using Proposition 10, the event β can be removed from the system. Thus, EFSMs E_1 and E_2 are replaced by $E_{1|\Sigma_1 \setminus \{\beta\}}$ and $E_{2|\Sigma_2 \setminus \{\beta\}}$ shown in Fig. 11. Note that, although event α also appears only on selfloops in the entire system, its update is not a pure guard, and consequently this event cannot be removed.

Using Propositions 9 and 10, it is possible to remove an event from the system. The following results make it possible to combine some events and replace them by a single event.



Event	Update
α	$y' = y + 1$
β	$y = 1$
γ	$y' = x$
σ	$y = 2$

Fig. 11 Example of selfloop removal.

Proposition 11 (Event Merging) *Let $\mathscr{E} = \{E_1, \dots, E_n\}$ be a normalised EFSM system with $E_i = \langle \Sigma_i, Q_i, \to_i, Q^\circ_\omega, Q^\omega_i \rangle$, let $E_k \in \mathscr{E}$, and let $\rho\colon \Sigma_\mathscr{E} \to \Sigma'$ be a renaming such that the following conditions hold for all $\sigma_1, \sigma_2 \in \Sigma_\mathscr{E}$ with $\rho(\sigma_1) = \rho(\sigma_2)$:*

(i) $\Delta_\mathscr{E}(\sigma_1) = \Delta_\mathscr{E}(\sigma_2)$;

(ii) *for all $i \neq k$, it holds that $\sigma_1 \in \Sigma_i$ if and only if $\sigma_2 \in \Sigma_i$, and for all $x, y \in Q_i$ it holds that $x \xrightarrow{\sigma_1:\Delta_\mathscr{E}(\sigma_1)}_i y$ if and only if $x \xrightarrow{\sigma_2:\Delta_\mathscr{E}(\sigma_2)}_i y$.*

Then $\mathscr{E}$ is nonblocking if and only if $\rho(\mathscr{E})$ is nonblocking.

By Proposition 11, all events with the same update can be merged and replaced by the same event if they appear on transitions with the same source and target states in all the EFSMs of the system except for one EFSM E_k. A proof is given in Appendix B.5.

One of the applications of event merging, Proposition 11, is after variable unfolding as events introduced by variable unfolding appear on different transitions only in the variable EFSM. For example, after unfolding the variable v_2 in Fig. 6 in Section 3, event s_2 is replaced by two new events $(s_2, 1, 0)$ and $(s_2, 2, 1)$ both with the same update *true*. Using Proposition 11, these events can be merged back into s_2. Therefore, there is no need to introduce new events in the first place, and only s_2 is used in Fig. 6.

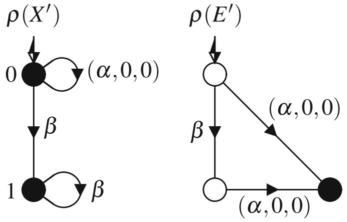
Example 10 Consider again the EFSMs E' and X' in Fig. 10. The updates of events $(\beta, 0, 1)$ and $(\beta, 1, 1)$ are the same, and these events appear on different transitions only in the EFSM X'. Thus, the renaming ρ is introduced such that $\rho((\beta, 0, 1)) = \rho((\beta, 1, 1)) = \beta$ and $\rho(\sigma) = \sigma$ for all $\sigma \notin \{(\beta, 0, 1), (\beta, 1, 1)\}$. The resulting simplified EFSMs $\rho(E')$ and $\rho(X')$ are shown in Fig. 12.

Proposition 12 (Update Merging) *Let $\mathscr{E} = \{E_1, \dots, E_n\}$ be a normalised EFSM system with $E_i = \langle \Sigma_i, Q_i, \to_i, Q^\circ_\omega, Q^\omega_i \rangle$. Let ρ be a renaming such that the following conditions hold for all $\sigma_1, \sigma_2 \in \Sigma_\mathscr{E}$ with $\rho(\sigma_1) = \rho(\sigma_2)$:*

(i) *$vars'(\Delta_\mathscr{E}(\sigma_1)) = vars'(\Delta_\mathscr{E}(\sigma_2))$,*

(ii) *for all $i = 1, \dots, n$ it holds that $\sigma_1 \in \Sigma_i$ if and only if $\sigma_2 \in \Sigma_i$, and for all $x, y \in Q_i$ it holds that $x \xrightarrow{\sigma_1:\Delta_\mathscr{E}(\sigma_1)}_i y$ if and only if $x \xrightarrow{\sigma_2:\Delta_\mathscr{E}(\sigma_2)}_i y$*

Further let $\mathscr{F} = \{F_1, \dots, F_n\}$ such that $F_i = \langle \rho(\Sigma_i), Q_i, \to_i^F, Q_i^\circ, Q_i^\omega \rangle$ where $\to_i^F = \{(x, \rho(\sigma), \Delta_\mathscr{F}(\rho(\sigma)), y) \mid x \xrightarrow{\sigma:\Delta_\mathscr{E}(\sigma)} y\}$ and $\Delta_\mathscr{F}(\mu) \equiv \bigvee_{\sigma \in \rho^{-1}(\mu)} \Delta_\mathscr{E}(\sigma)$ for all $\mu \in \Sigma_\mathscr{F}$. Then $\mathscr{E}$ is nonblocking if and only if $\mathscr{F}$ is nonblocking.



Event	Update
$(\alpha,0,0)$	$y' = 1$
β	$y = 0$

Fig. 12 Example of event merging

By Proposition 12, two events with the same next-state variables in their updates that appear on transitions with the same source and target states in the entire system can be replaced by a single new event. The update of the new event is the disjunction of the updates of the replaced events. The condition on the next-state variables ensures that the set of unchanged variables, i.e., variables not occurring as next-state variables, is preserved. A proof can be found in Appendix B.5.

Example 11 Consider the normalised EFSM system $\mathscr{E} = \{E_1, E_2\}$ in Fig. 13 with $\mathrm{dom}(x) = \mathrm{dom}(y) = \{0, 1\}$. Events α_1 and α_2 appear on the same transitions throughout $\mathscr{E}$, and $\mathrm{vars}'_{\mathscr{E}}(\alpha_1) = \mathrm{vars}'_{\mathscr{E}}(\alpha_2) = \emptyset$. In order to use Proposition 12, the new event α and the renaming ρ are introduced such that $\rho(\alpha_1) = \rho(\alpha_2) = \alpha$, $\rho(\beta) = \beta$, and $\rho(\gamma) = \gamma$. The update of the merged event α is $y = 0 \vee y = 1 \Leftrightarrow_V \mathit{true}$, and as a result the EFSM E_1 can be replaced by F_1 in Fig. 13.

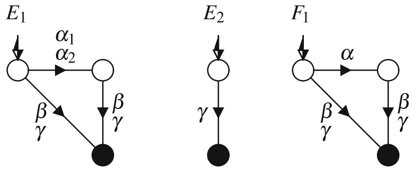
6 Algorithm

This section applies the results from the previous sections to give an algorithm for compositional nonblocking verification of EFSM systems. An overview of the approach is shown as Algorithm 1. Given an EFSM system, the algorithm repeatedly unfolds variables, composes EFSMs, and applies abstraction to the resultant EFSMs. While doing this, it maintains an *event tree* data structure to keep track of the events and their updates, and the renamings generated by partial unfolding.

Algorithm 1 EFSM-based compositional verification

```
 1: input $\mathscr{E} = \{E_1, E_2, \ldots, E_n\}$, $V = \mathrm{vars}(\mathscr{E})$
 2: $\mathscr{E} \leftarrow \mathrm{normalise}(\mathscr{E})$
 3: while $|\mathscr{E}| > 1 \vee |V| > 0$ do
 4:     $\langle V_c, \mathscr{E}_c \rangle \leftarrow \mathrm{selectCandidate}(\mathscr{E})$
 5:     if $V_c \neq \emptyset$ then
 6:         $v \leftarrow \mathrm{selectVariable}(V_c)$
 7:         $V \leftarrow V \setminus \{v\}$
 8:         $E \leftarrow \mathrm{unfold}(v)$
 9:     else
10:         $\mathscr{E} \leftarrow \mathscr{E} \setminus \mathscr{E}_c$
11:         $E \leftarrow \mathrm{synchronise}(\mathscr{E}_c)$
12:     end if
13:     $\mathrm{removeEvents}(\mathscr{E} \cup \{E\})$
14:     $\Upsilon \leftarrow \mathrm{getLocalEvents}(E, \mathscr{E})$
15:     $E \leftarrow \mathrm{simplify}(E, \Upsilon)$
16:     $\mathrm{removeEvents}(\mathscr{E} \cup \{E\})$
17:     $\mathscr{E} \leftarrow \mathscr{E} \cup \{E\}$
18: end while
19: $\mathrm{monolithicVerification}(\mathscr{E})$
```

The first step of Algorithm 1 is to normalise the system. The normalise() procedure in line 2 performs the normalisation and initialises the event tree, which at the beginning only links each event to its unique update. Then the main loop in lines 3–18 repeatedly unfolds variables or composes EFSMs, and simplifies EFSMs, until only a single EFSM is left.



Event	Update
α_1	$y = 0$
α_2	$y = 1$
β	$y' = x$
γ	$y = 1$
α	*true*

Fig. 13 Example of update merging

In each iteration, the selectCandidate() procedure in line 4 heuristically selects a *candidate* representing a subsystem to simplify, which consists of a set V_c of variables and a set $\mathscr{E}_c$ of EFSMs, the composition or unfolding of which is predicted to have potential for the most simplification. If the selected subsystem contains variables, then the selectVariable() procedure in line 6 heuristically identifies the best variable to unfold, which is then removed from the system and unfolded by the unfold() procedure in lines 7–8. If the selected candidate contains no variables, then it consists of only EFSMs, which are removed from the system and composed in lines 10–11. In both cases, the EFSM E resulting from unfolding or composition is sent for abstraction in lines 13–17.

Abstraction starts by calling the procedure removeEvents() in line 13, which applies update simplification (Proposition 7), removes events with *false* updates (Proposition 9), removes selfloop-only events (Proposition 10), and merges events (Propsosition 11 and 12). These steps can change both the EFSM E and the remaining components of $\mathscr{E}$, while at the same time updating the event tree. Afterwards, the simplify() procedure called in line 15 computes a conflict-preserving abstraction of E according to Proposition 5. As abstraction may change or remove transitions, it may result that some events only appear as selfloops or that there are groups of events on transitions with the same source and target states. Therefore, the removeEvents() procedure is applied again to the abstracted EFSM in an attempt to remove more events.

The loop terminates when $\mathscr{E}$ contains only one EFSM and all the variables are unfolded. The final EFSM can be considered as an FSM as there are no more variables in the system, so it is passed to standard FSM-based nonblocking verification in line 19. In the following, each of the procedures mentioned above is explained in more detail.

The selectCandidate() method in line 4 uses the following steps to select the most promising candidate, which consists of a set V_c of variables and a set $\mathscr{E}_c$ of EFSMs to be composed or unfolded.

(i) The first step is to search the system for variables that appear only as (primed) next-state variables or only as (unprimed) current-state variables in the entire system. If a variable only appears as next-state variable then all states in the variable EFSM are bisimilar (Milner 1989), so the variable EFSM can immediately be simplified to a single-location EFSM. If a variable only appears as current-state variable, then it never changes its value. Then all locations except the initial locations are unreachable in the variable EFSM, which again can be simplified to a single-location EFSM. If the system contains variables that only appear as current-state or only as next-state variables, then the selectCandidate() procedure returns all these variables in V_c, while $\mathscr{E}_c$ is an empty set.

(ii) The second step is to search for *local variables*, i.e., for variables that appear in the updates of only one EFSM. As the system is normalised, events with a local variable in their updates appear in only one EFSM. Unfolding local variables simplifies their updates, which may result in some updates becoming *true*, increasing the possibility of hiding and abstractions in later steps. Yet, after unfolding a local variable, its events in the EFSM are shared with the variable EFSM.

If the system contains local variables, the selectCandidate() procedure returns the local variables in V_c, while $\mathscr{E}_c$ is an empty set. Subsequently, one of these variables will be unfolded in line 8. To exploit the benefit of this unfolding, the selectCandidate() procedure also ensures that, in the next iteration of the main loop after unfolding a local variable and simplifying the resultant EFSM, the next candidate only consists of the variable EFSM and the EFSM that has the local variable.

(iii) If there are no variables that appear only primed or only unprimed, and no local variables, the final step is to use a strategy called mustL (Flordal and Malik 2009), which tries to improve the potential of abstraction by making events local. For each event σ, all EFSMs with σ in the alphabet and all variables in the update of σ are selected, so that σ becomes a local event if the selected EFSMs and variables were to be composed. This gives several candidates, one for each event, and the following heuristics are used to select the best candidate among them.

minF selects the candidate with the smallest number of other EFSMs and variables linked via events to the candidate's EFSMs and variables (Mohajerani et al. 2014). This heuristic attempts to minimise event sharing between the candidate and the rest of the system.

minS selects the candidate with the smallest estimated number of states in its synchronous composition. The number of states in the synchronous composition is estimated as the product of the sizes of the domains of the variables and the numbers of locations of the EFSMs, multiplied by the ratio of the number of events the candidate shares with the rest of the system over the total number of events of the candidate (Flordal and Malik 2009).

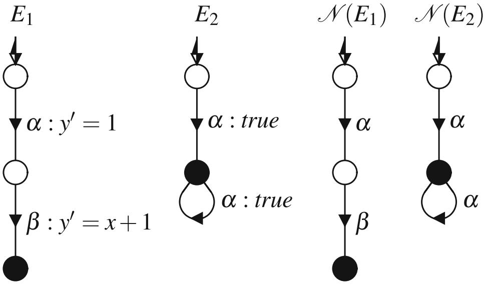
The selectCandidate() procedure first employs the minF heuristic, and if minF gives equal preference to two candidates, then the minS heuristic is used to break the tie.

If the candidate returned by the selectCandidate() procedure contains variables, only one of these variables will be unfolded. Line 6 calls the selectVariable() procedure, which uses another set of heuristics to identify the most promising variable:

maxE selects a variable that appears in the update of the largest number of events.
maxS selects a variable that appears in the largest number of selfloops.
minD selects a variable with the smallest domain.

The maxE heuristic enables more update simplification, which increases the chance of updates becoming *true* and thus the possibility of hiding. As the contributions of the conflict equivalence abstraction methods are high in simplifying the system, and they are greatly dependent on hiding, maxE is the first heuristic that is applied. If maxE gives equal preference to two variables, then the maxS heuristic is used to break the tie. maxS attempts to boost the performance of selfloop removal, which also may increase the performance of conflict equivalence abstraction. In the case that both maxE and maxS give equal preference to two variables, the minD heuristic is used in order to create the smallest possible variable EFSM.

Example 12 Consider the EFSM system $\mathscr{E} = \{E_1, E_2\}$ in Fig. 14. Normalisation produces $\mathscr{N}(E_1)$ and $\mathscr{N}(E_2)$ with the updates shown in the table. As variable x only appears



Event	Update
α	$y' = 1$
β	$y' = x+1$

Presentation of the table as event tree:

α	β
$y' = 1$	$y' = x+1$

Fig. 14 Normalisation and initial event tree

unprimed and variable y only appears primed in the updates, the selectCandidate() procedure returns $V_c = \{x, y\}$ and $\mathcal{E}_c = \emptyset$ as the selected candidate. As this candidate has variables, next the selectVariable() procedure is called in line 6 to select a variable, which starts by evaluating the maxE heuristic. In this case, x only occurs in the update of β, while y occurs in the updates of both α and β, so y is selected for unfolding.

According to Definition 22, partial unfolding by the unfold() procedure in line 8 generates for each event σ with the variable v in its update, several new events of the form (σ, a, b). Subsequently, all the components of the system need to be changed by replacing σ with these new events according to Definition 24. To improve performance, this replacement is done by modifying the event tree only. The new events (σ, a, b) are added with their updates to the event tree as children of the original event σ. Other EFSMs using σ remain unchanged, with the interpretation that any transition labelled σ represents transitions with all the descendants of σ stored in the event tree. In this way, the replacement of events and the associated blow-up in the number of transitions is postponed until the synchronous composition of EFSMs containing σ with the variable EFSM. Often, the replacement can be avoided altogether, as some of the new events can be removed or merged before they are needed in synchronous composition.

Example 13 Consider the normalised EFSM system $\mathcal{E} = \{\mathcal{N}(E_1), \mathcal{N}(E_2)\}$ in Fig. 14 with $\text{dom}(x) = \{0, 1\}$, $x_{\circ} = 0$, $\text{dom}(y) = \{1, 2\}$, and $y_{\circ} = 1$. Normalisation produces $\mathcal{N}(E_1)$ and $\mathcal{N}(E_2)$ with the updates stored separately in an event tree that initially consists of two root nodes with the two events α and β and their updates, as shown in Fig. 14. Unfolding of the variable y produces events of the form (α, a, b) and (β, a, b), which are added as children of α and β to the event tree. Figure 15 shows the variable EFSM $U_{\mathcal{E}}(y)$ and the updated event tree. At this point, the EFSMs $\mathcal{N}(E_1)$ and $\mathcal{N}(E_2)$ are left unchanged.

The removeEvents() procedure called in line 13 attempts to simplify the EFSMs and the event tree as much as possible before proceeding further with simplification. It first simplifies all the updates according to Proposition 7 and removes all events with *false* updates according to Proposition 9. Next, selfloop removal is applied, which removes events that have pure guards as updates and that only appear as selfloops in the entire system (Proposition 10). Afterwards, the removeEvents() procedure merges events and updates using Propostions 11 and 12. An exact implementation of these propositions requires a search of transitions of all the EFSMs. To improve performance, the removeEvents() uses the event tree. Only events that have the same parent and no children in the event tree are considered

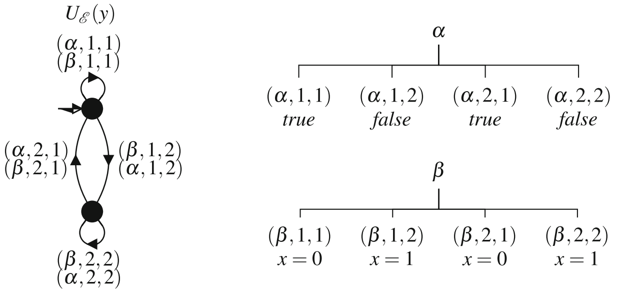


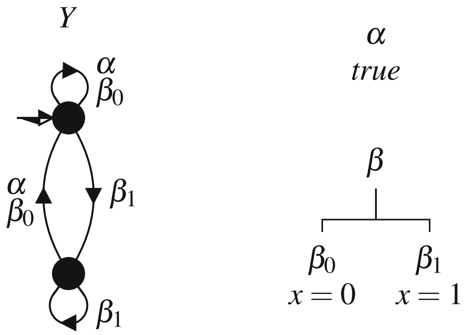
Fig. 15 Result of unfolding y and associated event tree

for merging. By the construction of the event tree, the children of an event σ are implicitly present in all EFSMs containing the parent event σ, but they appear on exactly the same transitions as the parent event, so these EFSMs do not need to be checked to determine whether merging is possible according to Propositions 11 and 12. More precisely, the removeEvents() procedure first searches for childless events with the same parent and the same next-state variables in their updates. Each group of such events becomes a *merge candidate*. Within a merge candidate, all events with the same update p are replaced by a single event with update p (Proposition 11), and the remaining events that appear on transitions with the same source and target states are replaced by a new event with an update that is the disjunction of the updates of the replaced events (Proposition 12).

Example 14 Consider the EFSM system $\mathscr{E} = \{\mathscr{N}(E_1), \mathscr{N}(E_2), U_{\mathscr{E}}(y)\}$ with the EFSMs shown in Figs. 14 and 15, where $\mathrm{dom}(x) = \{0, 1\}$ and $x_\circ = 0$. First, the removeEvents() procedure removes events $(\alpha, 1, 2)$ and $(\alpha, 2, 2)$ as their updates are *false*. Next, events $(\alpha, 1, 1)$ and $(\alpha, 2, 1)$ both have the parent α and the update *true*. These events can be merged according to Proposition 11, and as they are the only children of α they are replaced by their parent α with update *true*. Further, the events $(\beta, 1, 1)$ and $(\beta, 2, 1)$, and $(\beta, 1, 2)$ and $(\beta, 2, 2)$ have the same update and the same parent. The removeEvents() procedure merges events $(\beta, 1, 1)$ and $(\beta, 2, 1)$ into a new event β_0, and merges $(\beta, 1, 2)$ and $(\beta, 2, 2)$ into a new event β_1. Then β is assigned as the parent of β_0 and β_1 in the event tree. Figure 16 shows the EFSM Y that results from $U_{\mathscr{E}}(y)$ and the event tree after these modifications.

After event removal and update merging, the simplify() procedure called in line 15 attempts to simplify the EFSM E based on conflict equivalence. First, the getLocalEvents()

Fig. 16 Result of the removeEvents() procedure applied to $U_{\mathscr{E}}(y)$



procedure called in line 14 computes the set Υ of events to be hidden. According to Proposition 5, these are events with *true* update that only appear in E. Then the EFSM is converted to FSM form while hiding these events. As the updates are stored separately from the EFSM E in the event tree, E is simply considered as an FSM and the events in Υ are replaced by τ. Then the conflict preserving abstraction methods (Flordal and Malik 2009) are applied to this FSM. Using the event tree, the resultant FSM can again be considered as an EFSM.

The synchronise() procedure, called in line 11, performs normalised synchronous composition, ignoring updates, and taking the event relationships in the event tree into account. If the set $\mathscr{E}_c$ of EFSMs to be composed contains an EFSM containing a parent event and an EFSM containing its children, then the procedure replaces all occurrences of the parent event with its children while computing the normalised synchronous composition.

Example 15 Consider the normalised EFSM system $\mathscr{E} = \{\mathscr{N}(E_1), \mathscr{N}(E_2), Y\}$ with the EFSMs shown in Figs.14 and 16, and assume that the EFSMs Y and $\mathscr{N}(E_1)$ are to be composed. During synchronous composition, based on the event tree, event β in $\mathscr{N}(E_1)$ is replaced by β_0 and β_1, resulting in EFSM E_1', while Y remains unchanged. Figure 17 shows the EFSM E_1' and the result $E_1'||Y$ of normalised synchronous composition.

7 Experimental results

The EFSM-based compositional nonblocking verification has been implemented in the discrete event systems tool Supremica (Åkesson et al. 2006). It has been tested on some large EFSM models and compared to an FSM-based compositional algorithm (Malik and Leduc 2013) and a BDD-based algorithm (Vahidi 2004). This section shows the results of the experiments.

The following list gives an overview of the test cases used to compare the algorithms. These include complex industrial models and case studies, and some scalable models with very large state spaces.

pml3 A system consisting of three parallel manufacturing lines each processing workpieces of a particular type (Mohajerani et al. 2014). Each line has 49 serially connected machines. The parallel lines share 50 buffers with capacity 2 or 3.

prime-sieve A distributed version of the *Sieve of Eratosthenes* for generating prime numbers (Mohajerani et al. 2013b). The models considered here contain filters for the first 4,

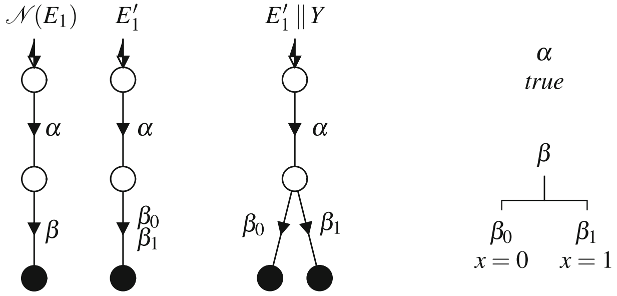


Fig. 17 Synchronous composition of $\mathscr{N}(E_1)$ and Y using event tree

6, or 8 prime numbers, enabling them to find prime numbers less or equal to 120, 288, or 528, respectively. **prime-sieve4b** is a faulty blocking version of the program, while the others are correct and nonblocking.

production-cell A model of part of a metal-processing plant consisting of seven manufacturing devices (Fabian et al. 2014).

profisafe-ihost-b Part of the PROFIsafe protocol for fail-safe communication (Malik and Mühlfeld 2003). Considered here are two EFSM models of a blocking version of the PROFIsafe input-host with sequence numbers ranging from 0 to 127 or 255.

psl An assembly cell for toy cars, which are built up from seven parts (Parsaeian 2014). The cell has three robots, which pick up parts from two intakes, assemble them in a fixture, and thereafter place the complete car at an outlet. **psl-big** is a model of the uncontrolled plant, **psl-n** is the nonblocking system with supervisor updates added to the EFSMs. **psl-b** is a blocking version obtained from **psl-n** by removing some of the supervisor updates, **psl-restart** is an uncontrolled plant with additional transitions to facilitate resynchronisation of the system.

round-robin EFSM model of a token-passing resource allocation protocol with a ring of 300 or 400 processes (Graf and Steffen 1990).

tline A scalable version of the transfer line with rework cycles (Teixeira et al. 2013). The system consists of 500 serially connected cells linked by buffers. Each cell has three machines and a test unit that can reject workpieces up to 3 or 4 times.

Table 1 shows experimental results for nonblocking verification of the above models using the EFSM-based compositional nonblocking verification algorithm proposed in Section 6 and two other methods. The table shows for each model the number of EFSMs ($|\mathscr{E}|$), the number of variables ($|V|$), the size of the largest variable domain (|dom|), whether or not the model is nonblocking (Nbl), and the performance data for the different algorithms. The experiments were run on a standard desktop computer using a single core 3.3 GHz CPU and not more than 2 GB of RAM. Runtime was limited to 20 minutes: if this time was exceeded, or the process ran over the limit of 2 GB RAM, the attempt was aborted and the table entries left blank.

The EFSM column shows the runtimes of the EFSM-based compositional nonblocking verification and the total number of EFSM transitions encountered during the run. The implementation proceeds as described in Section 6 and selects the subsystems to compose and the variables to unfold following the heuristics in the order presented. That is, candidates for composition are selected according to the minF heuristic, and if this gives the same priority for two candidates, then the minS heuristic is used to break the tie. Likewise, the order for the variable selection heuristics is first maxE, then maxS, and finally minD. The number of transitions shown in the table is the sum of the transition numbers of the EFSMs E encountered in line 13 of Algorithm 1, before abstraction, plus the number of transitions of the final EFSM in line 19.

For comparison, the table also contains runtimes for the following two alternative algorithms from previous work, which are also implemented in Supremica.

FSM The FSM-based compositional algorithm converts the EFSM model to a set of FSMs and then applies the compositional algorithm (Flordal and Malik 2009). The EFSM model is *modularly* flattened by creating a collection of *location FSMs* and *variable FSMs* (Mohajerani et al. 2013a). Location FSMs use the EFSM locations as states but replace the updates with events. The variable FSMs use the domain of a variable as their states space and keep track of the value of that variable. Differently from Section 5.4

Table 1 Experimental results for different nonblocking verification algorithms

Model					Flattening+FSM			BDD		EFSM	
Name	$\lvert\mathscr{E}\rvert$	$\lvert V\rvert$	|dom|	Nbl	Flatten	Verify	Transitions	Time	Nodes	Time	Transitions
pml3 ⟨2⟩	153	200	4	yes	1.5 s	2.0 s	59,132			2.9 s	23,594
pml3 ⟨3⟩	153	200	4	yes	2.8 s	25.2 s	553,616			6.3 s	116,691
prime-sieve4b	6	10	121	no	6.0 s	2.5 s	371,454	3.5 s	780,241	2.8 s	203,060
prime-sieve4	6	10	121	yes	5.9 s	2.3 s	405,400	5.0 s	712,594	2.7 s	200,438
prime-sieve6	8	14	289	yes	130.4 s	23.0 s	3,291,713	74.2 s	6,503,464	19.6 s	1,598,433
prime-sieve8	10	18	529	yes						137.7 s	6,945,315
production-cell	17	32	3	no	0.3 s	1.1 s	38,594	0.4 s	46,538	0.8 s	4,567
profisafe-ihost-b ⟨127⟩	4	17	128	no	345.8 s	206.8 s	5,584,320	5.5 s	4,034,794	28.6 s	1,444,584
profisafe-ihost-b ⟨255⟩	4	17	256	no						101.2 s	3,079,912
psl-big	1	37	14	no	0.5 s	0.5 s	5,299	1.6 s	92,356	1.5 s	10,217
psl-b	1	37	14	no	54.1 s	349.4 s	380,847	35.0 s	5,448,127	5.5 s	190,695
psl-n	1	37	14	yes	54.6 s	370.1 s	458,162	38.5 s	5,517,392	5.8 s	229,080
psl-restart	1	37	14	no	0.6 s	3.5 s	129,490	1.1 s	120,690	4.2 s	182,946
round-robin ⟨300⟩	901	1	300	no	1.5 s	87.3 s	14,720			69.7 s	11,119
round-robin ⟨400⟩	1201	1	400	no	2.0 s	210.6 s	19,620			166.5 s	14,819
tline ⟨3⟩	1501	3001	4	yes	27.2 s	56.1 s	1,379,328			98.6 s	2,841,834
tline-b ⟨3⟩	1501	3001	4	no	27.2 s	851.6 s	3,460,482			99.3 s	2,843,887
tline ⟨4⟩	1501	3001	5	yes	43.5 s	109.2 s	5,312,091			156.2 s	10,410,971
tline-b ⟨4⟩	1501	3001	5	no	44.4 s	898.1 s	12,029,935			157.8 s	10,419,370

above, the flattened FSM system has events of the form $(\sigma; \hat{v}; \hat{w})$ for each update p of event σ in EFSM E and all valuations $\hat{v} \in \mathrm{dom}(\mathrm{vars}(p))$ and $\hat{w} \in \mathrm{dom}(\mathrm{vars}'(p))$ such that $p(\hat{v}, \hat{w}) = \mathbf{T}$. In the worst case, the number of events created for an update is the product of the sizes of the domains of its variables. Table 1 shows the total number of transitions encountered during verification, obtained in the same way as for the EFSM column, and in two further columns the times spent to flatten the system (Flatten) and to verify the flattened FSM system (Verify). The total runtime is the sum of these two columns.

BDD The BDD-based algorithm converts the EFSM model to a symbolic representation in the form of Binary Decision Diagrams (BDDs) (Bryant 1992) and explores the full state space symbolically (McMillan 1993). The implementation includes several performance improvements, most importantly an initial variable ordering based on the FORCE heuristics (Aloul et al. 2003) and a disjunctive partitioning of the transition relation in a form optimised for discrete event systems (Vahidi 2004). Table 1 shows the total runtime of BDD-based verification and the total number of BDD nodes encountered. BDD nodes require a similar amount of memory as transitions, so that their number gives a measure of the memory requirements similar to the numbers of transition encountered by the other two algorithms.

Table 1 shows that the EFSM-based compositional algorithms successfully verifies all the examples in this experiment, in most cases faster than the two other algorithms.

The transition numbers encountered by the EFSM-based algorithm are usually smaller than those encountered by the FSM-based algorithm, as is to be expected from the deferred unfolding and event tree techniques.

However, the EFSM algorithm needs to perform symbolic computation and can take more time even with fewer numbers. For examples like **pml3** ⟨2⟩, **production-cell**, and **round-robin** the runtimes of the FSM-based and EFSM-based algorithms are similar. These models have got simple updates, so that the number of events in the flattened FSM system is small and the flattening times are small, and as a result, the FSM-based and EFSM-based algorithms verify the models in similar ways.

This effect can also be observed with the **psl** models. The models **psl-b** and **psl-n** are obtained by calculating a supervisor and attaching some or all its updates to **psl-big**, resulting in EFSMs with a large number of extremely complicated updates. As it can be observed from the table, the **psl-big** and **psl-restart** models, which have much fewer and simpler updates, can be flattened significantly faster than **psl-b** and **psl-n**. In these cases, the FSM-based algorithm encounters fewer transitions than the EFSM-based algorithm, suggesting that the FSM-based method can perform more accurate simplification for these simple models. For **psl-b** and **psl-n**, the large large number of events in the flattened FSM system causes the FSM-based compositional algorithm to perform poorly, while both the EFSM-based compositional and the BDD algorithm cope well with the complicated updates.

The BDD algorithm is not impeded by complicated update formulas, as these only cause a moderate increase in the complexity of the symbolic model. However, it is limited by the size and search depth of the state space, and fails to verify large scaled-up models such as **pml**, **round-robin** and **tline**.

The nonblocking verification algorithms have also been applied to a scalable version of the *manufacturing system* example in Section 3. The system consists of M serially connected cells linked by conveyor belts. Each cell m for $1 \leq m \leq M$ has two conveyor belt sections CB_1^m and CB_2^m and two machines M_1^m and M_2^m. Initially workpieces are picked up by the conveyor belt CB_1^1 to enter cell 1, and completed workpieces from M_1^m or M_2^m in cell m are placed on the next conveyor CB_1^{m+1}. Fig. 18 shows the runtimes for instances with 1–1000 serially connected cells and fixed conveyor belt capacity $N = 10$, and Fig. 19 shows the runtimes for instances with conveyor belt capacity 2–100 and a fixed number of 100 serially connected cells.

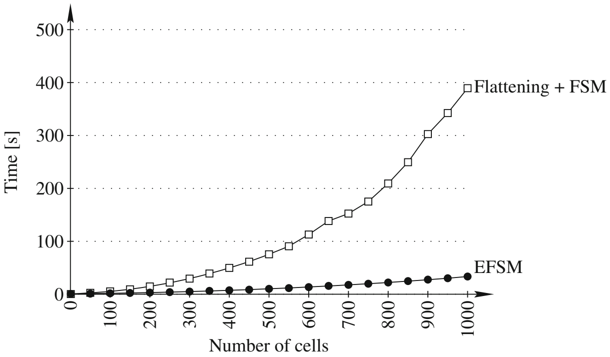


Fig. 18 Runtimes for manufacturing systems with conveyor capacity $N = 10$ and increasing number of cells

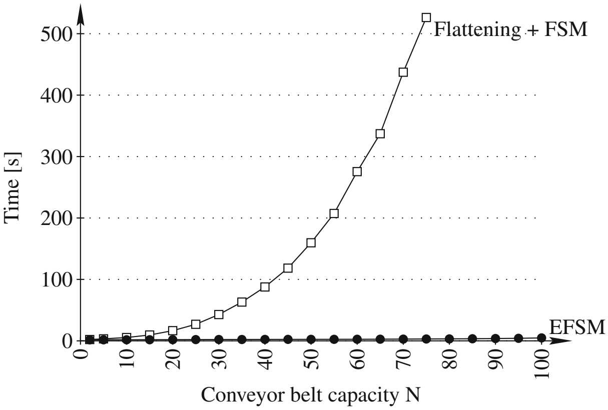


Fig. 19 Runtimes for manufacturing systems with 100 cells and increasing conveyor belt capacity *N*

The EFSM-based compositional algorithm successfully reveals the blocking conditions of manufacturing systems with up to 1000 serially connected cells and conveyor belt capacity up to 100. However, the runtime does not grow linearly in the number of connected cells because of the complexity of the MustL and MinF heuristics. The preselection heuristic MustL produces up to $|\Sigma|$ candidates, and the evaluation of MinF takes time proportional to

Table 2 Runtimes with different composition candidate and variable selection heuristics

	minF			minS		
	maxE	maxS	minD	maxE	maxS	minD
pml3 ⟨2⟩	2.88 s	3.10 s	3.01 s			
pml3 ⟨3⟩	6.34 s	33.99 s	6.42 s			
prime-sieve4b	2.80 s	2.82 s	2.84 s	2.82 s	2.81 s	2.80 s
prime-sieve4	2.67 s	2.68 s	2.68 s	2.68 s	2.69 s	2.70 s
prime-sieve6	19.64 s	19.95 s	19.78 s	19.88 s	19.72 s	19.71 s
prime-sieve8	137.70 s	136.51 s	137.86 s	137.43 s	137.17 s	137.15 s
production-cell	0.84 s	0.85 s	0.86 s	0.74 s	0.75 s	0.74 s
profisafe-ihost-b ⟨127⟩	28.64 s	29.06 s	29.88 s	23.69 s	23.11 s	23.54 s
profisafe-ihost-b ⟨255⟩	101.19 s	102.04 s	97.70 s			
psl-big	1.51 s	1.47 s	1.39 s	1.46 s	1.45 s	1.61 s
psl-b	5.49 s	8.14 s	84.10 s	9.42 s	9.46 s	112.44 s
psl-n	5.82 s	8.01 s	88.44 s	10.42 s	8.98 s	115.39 s
psl-restart	4.15 s	4.20 s	46.67 s	4.25 s	4.14 s	32.92 s
round-robin ⟨300⟩	69.72 s	68.97 s	69.74 s			
round-robin ⟨400⟩	166.48 s	166.95 s	166.57 s			
tline ⟨3⟩	98.56 s	96.69 s	97.06 s			
tline-b ⟨3⟩	99.31 s	97.17 s	97.75 s			
tline ⟨4⟩	156.22 s	155.91 s	155.93 s			
tline-b ⟨4⟩	157.85 s	157.52 s	156.56 s			

the number of events of the candidate. Therefore, the complexity of the component selection heuristic is quadratic in the number of events. The FSM-based algorithm has more events to process and therefore suffers more from this effect, and in addition most of its runtime is taken up by the flattening process. The difference is more noticeable when the conveyor belt capacity increases than when the number of cells increases, because the number of events grows quadratically in the size of the domain of the variable N and only linearly in the number of cells. The BDD algorithm cannot handle more than 60 serially connected cells and therefore is not shown in these figures.

Table 2 shows the runtimes of the EFSM-based compositional nonblocking verification algorithm using different heuristics for the selection of composition candidates and variables. In this experiment, the standard ordering of the heuristics is changed such that the indicated heuristic is used first. In the maxS columns, the order of variable selection heuristics is first maxS, then maxE, and finally minD; and in the minD columns, the order is first minD, then maxE, and finally maxS.

The results suggest that, in many cases the variable ordering heuristics have little or no effect on the runtime. On the other hand, while the algorithm fails to verify some of the models using minS, it successfully verifies all the examples using minF. It appears that the minF heuristic is better at increasing the number of local events and thus the possibility of abstraction.

8 Conclusions

A general framework for compositional nonblocking verification of extended finite-state machines (EFSMs) has been presented, which supports the verification of large models consisting of several EFSMs that interact both via shared events and shared variables. Normalisation is introduced, which makes it possible to unfold arbitrary variables and apply abstraction methods developed previously for ordinary finite-state machines to EFSMs, without the need to flatten the system and the associated overhead. Various methods of abstraction are presented that simplify individual system components while preserving the nonblocking property of the whole system.

These results are combined in an algorithm for compositional nonblocking verification of EFSM systems. This algorithm gradually composes the system and applies conflict equivalence abstraction to the components, unfolds variables, and removes events if possible. The algorithm has been implemented and its performance compared with two other well-developed algorithms. The experimental results suggest that the EFSM-based algorithm can outperform FSM-based and BDD-based methods for large systems with complex update formulas on their transitions.

In future work, the authors would like to improve the algorithm using better symbolic abstractions and considering special properties of updates. Another area of interest is to extend the method and apply it to the compositional synthesis of supervisors for EFSM systems.

Appendix: A Normalisation

This appendix contains the proofs of the propositions concerning normalisation presented in Section 4. First Proposition 1 confirms that EFSMs obtained by normalised synchronous composition in Definition 14 and by standard synchronous composition in Definition 9 produce identical flattened FSMs. Next, Proposition 2 confirms that the structure of an EFSM system is preserved after normalisation of individual components. Finally, Proposition 3 confirms that the normalised system is identical to the original system.

Proposition 1 *Let $\mathcal{E}$ be a normalised EFSM system. Then $U(\|\mathcal{E}) = U(\dot{\|}\mathcal{E})$.*

Proof It follows from Definitions 9, 10, and 14 that $U(\|\mathcal{E})$ and $U(\dot{\|}\mathcal{E})$ have the same event and state sets, including initial and marked states. It remains to be shown that they also have the same transitions.

To show this, write $\mathcal{E} = \{E_1, \ldots, E_n\}$, and $\Sigma_i = \Sigma_{E_i}$ and $\Delta_i = \Delta_{E_i}$ for $1 \leq i \leq n$.

First let $(x_1, \ldots, x_n, \hat{v}) \overset{\sigma}{\to} (y_1, \ldots, y_n, \hat{w})$ in $U(\|\mathcal{E})$. By Definition 10 this means $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\|\mathcal{E}$ such that $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$, where $p \equiv \bigwedge_{i:\sigma\in\Sigma_i} \Delta_{\mathcal{E}}(\sigma)$ by Definition 9 and by the fact that $\mathcal{E}$ is normalised. Consider two cases for each E_i: either $\sigma \in \Sigma_i$ or $\sigma \notin \Sigma_i$. If $\sigma \in \Sigma_i$, then $x_i \xrightarrow{\sigma:\Delta_{\mathcal{E}}(\sigma)} y_i$ in E_i. If $\sigma \notin \Sigma_i$, then $x_i = y_i$. Then $(x_1, \ldots, x_n) \xrightarrow{\sigma:\Delta_{\mathcal{E}}(\sigma)} (y_1, \ldots, y_n)$ in $\dot{\|}\mathcal{E}$ by Definition 14, and as $\Xi_V(\bigwedge_{i:\sigma\in\Sigma_i} \Delta_{\mathcal{E}}(\sigma))(\hat{v}, \hat{w}) = \Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$, it holds that $\Xi_V(\Delta_{\mathcal{E}}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$. Thus, $(x_1, \ldots, x_n, \hat{v}) \overset{\sigma}{\to} (y_1, \ldots, y_n, \hat{w})$ in $U(\dot{\|}\mathcal{E})$ by Definition 10.

Conversely, assume $(x_1, \ldots, x_n, \hat{v}) \overset{\sigma}{\to} (y_1, \ldots, y_n, \hat{w})$ in $U(\dot{\|}\mathcal{E})$. By Definition 10 this means $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\dot{\|}\mathcal{E}$ such that $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$. Consider two cases for each E_i: either $\sigma \in \Sigma_i$ or $\sigma \notin \Sigma_i$. If $\sigma \in \Sigma_i$, then by Definition 14 it follows that $x_i \xrightarrow{\sigma:p} y_i$ in E_i. If $\sigma \notin \Sigma_i$, then $x_i = y_i$. By Definition 9, it follows that $(x_1, \ldots, x_n) \xrightarrow{\sigma:\bigwedge_{i:\sigma\in\Sigma_i} p} (y_1, \ldots, y_n)$ in $\|\mathcal{E}$, and as $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$, it follows that $\Xi_V(\bigwedge_{i:\sigma\in\Sigma_i} p)(\hat{v}, \hat{w}) = \mathbf{T}$. Thus, $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma:p} (y_1, \ldots, y_n, \hat{w})$ in $U(\|\mathcal{E})$ by Definition 10. □

Proposition 2 *Let $\mathcal{E}$ and $\mathcal{F}$ be EFSM systems, and let $\rho : \Sigma_{\mathcal{F}} \to \Sigma_{\mathcal{E}}$ be a renaming, such that $\mathcal{E} = \{E_1, E_2, \ldots, E_n\}$ and $\mathcal{F} = \{F_1\, \rho^{-1}(E_2), \ldots, \rho^{-1}(E_n)\}$ and $\rho(F_1) = E_1$. Then $\rho(\|\mathcal{F}) = \|\mathcal{E}$.*

Proof Let

$$E = \|\mathcal{E} = E_1\|\cdots\|E_n\ ; \tag{24}$$

$$F = \|\mathcal{F} = F_1\|F_2\|\cdots\|F_n = F_1\|\rho^{-1}(E_2)\|\cdots\|\rho^{-1}(E_n)\ . \tag{25}$$

Clearly $\Sigma_E = \Sigma_{\mathcal{E}} = \rho(\Sigma_{\mathcal{F}}) = \rho(\Sigma_F)$, and from $\rho(F_1) = E_1$ it follows that E and $\rho(F)$ have the same state sets, including initial and marked states. It remains to be shown that E and $\rho(F)$ have the same transitions.

First assume $(x_1, x_2, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, y_2, \ldots, y_n)$ in $E = E_1\|\cdots\|E_n$. By Definition 9 it holds that $p = \bigwedge_{i:\sigma\in\Sigma_{E_i}} p_i$ where $x_i \xrightarrow{\sigma:p_i} y_i$ in E_i whenever $\sigma \in \Sigma_{E_i}$. Consider two cases.

- If $\sigma \in \Sigma_{E_1}$ then $x_1 \overset{\sigma:p_1}{\to} y_1$ in E_1. Since $\rho(F_1) = E_1$, there exists $\mu \in \Sigma_{F_1}$ such that $\rho(\mu) = \sigma$ and $x_1 \xrightarrow{\mu:p_1} y_1$ in F_1. Now consider two cases for each $2 \leq i \leq n$: either $\sigma \in \Sigma_{E_i}$ or $\sigma \notin \Sigma_{E_i}$. If $\sigma \in \Sigma_{E_i}$ then $x_i \xrightarrow{\sigma:p_i} y_i$ in E_i, and since $\rho(\mu) = \sigma$ it holds by Definition 16 that $x_i \xrightarrow{\mu:p_i} y_i$ in $\rho^{-1}(E_i)$. If $\sigma \notin \Sigma_{E_i}$ then

$\mu \notin \rho^{-1}(\Sigma_{E_i}) = \Sigma_{\rho^{-1}(E_i)}$ and $x_i = y_i$. Combining the above observations for all i, it follows by Definition 9 that $(x_1, x_2, \ldots, x_n) \xrightarrow{\mu:p} (y_1, y_2, \ldots, y_n)$ in F, which implies $(x_1, x_2, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, y_2, \ldots, y_n)$ in $\rho(F)$.

- If $\sigma \notin \Sigma_{E_1}$ then $x_1 = y_1$. As ρ is surjective by Definition 15, there exists $\mu \in \Sigma_{\mathscr{F}}$ such that $\rho(\mu) = \sigma$. Then since $\rho(\Sigma_{F_1}) = \Sigma_{E_1}$, it holds that $\mu \notin \Sigma_{F_1}$. Now consider two cases for each $2 \le i \le n$: either $\sigma \in \Sigma_{E_i}$ or $\sigma \notin \Sigma_{E_i}$. If $\sigma \in \Sigma_{E_i}$ then $x_i \xrightarrow{\sigma:p_i} y_i$ in E_i, and since $\rho(\mu) = \sigma$ it holds by Definition 16 that $x_i \xrightarrow{\mu:p_i} y_i$ in $\rho^{-1}(E_i)$. If $\sigma \notin \Sigma_{E_i}$ then $\mu \notin \rho^{-1}(\Sigma_{E_i}) = \Sigma_{\rho^{-1}(E_i)}$ and $x_i = y_i$. Combining the above observations for all i, it follows by Definition 9 that $(x_1, x_2, \ldots, x_n) \xrightarrow{\mu:p} (y_1, y_2, \ldots, y_n)$ in F, which implies $(x_1, x_2, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, y_2, \ldots, y_n)$ in $\rho(F)$.

Conversely assume $(x_1, x_2, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, y_2, \ldots, y_n)$ in $\rho(F)$. Then there exists $\mu \in \Sigma_{\mathscr{F}}$ such that $\rho(\mu) = \sigma$ and $(x_1, x_2, \ldots, x_n) \xrightarrow{\mu:p} (y_1, y_2, \ldots, y_n)$ in F. By Definition 9 it holds that $p \equiv \bigwedge_{i:\mu \in \Sigma_{F_i}} p_i$ where $x_i \xrightarrow{\mu:p_i} y_i$ in F_i whenever $\mu \in \Sigma_{F_i}$. Consider two cases for E_1:

- If $\mu \in \Sigma_{F_1}$ then $x_1 \xrightarrow{\mu:p_1} y_1$ in F_1. Since $\rho(\mu) = \sigma$, it follows that $x_1 \overset{\sigma:p_1}{\to} y_1$ in $\rho(F_1) = E_1$.
- If $\mu \notin \Sigma_{F_1}$ then $\sigma = \rho(\mu) \notin \rho(\Sigma_{F_1}) = \Sigma_{E_1}$ and $x_1 = y_1$.

Now consider two cases for each $2 \le i \le n$:

- If $\sigma \in \Sigma_{E_i}$ then $\mu \in \rho^{-1}(\Sigma_{E_i}) = \Sigma_{F_i}$ and therefore $x_i \xrightarrow{\mu:p_i} y_i$ in $F_i = \rho^{-1}(E_i)$. Since $\rho(\mu) = \sigma$, it holds by Definition 16 that $x_i \xrightarrow{\sigma:p_i} y_i$ in E_i.
- If $\sigma \notin \Sigma_{E_i}$ then $\mu \notin \rho^{-1}(\Sigma_{E_i}) = \Sigma_{F_i}$ and $x_i = y_i$.

Combining the above observations for $1 \le i \le n$, it follows by Definition 9 that $(x_1, x_2, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, y_2, \ldots, y_n)$ in $E_1 \| \cdots \| E_n = E$. □

Proposition 3 *Let $\mathscr{E}$ be an EFSM system such that each $E \in \mathscr{E}$ is normalised. Then $\|\mathscr{E} = \dot{\|}\mathscr{N}(\mathscr{E})$.*

Proof It follows from Definitions 9, 14, and 17 that $\|\mathscr{E}$ and $\dot{\|}\mathscr{N}(\mathscr{E})$ have the same event and state sets, including initial and marked states. It remains to be shown that they also have the same transitions.

To show this, write $\mathscr{E} = \{E_1, \ldots, E_n\}$, and $\Sigma_i = \Sigma_{E_i}$ and $\Delta_i = \Delta_{E_i}$ for $1 \le i \le n$.

First assume $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\|\mathscr{E}$. Then by Definitions 9 and 17, it holds that $p \equiv \bigwedge_{i:\sigma \in \Sigma_i} \Delta_i(\sigma) \equiv \Delta_{\mathscr{N}(\mathscr{E})}(\sigma)$. Consider two cases for each E_i: either $\sigma \in \Sigma_i$ or $\sigma \notin \Sigma_i$. If $\sigma \in \Sigma_i$, then $x_i \xrightarrow{\sigma:\Delta_i(\sigma)} y_i$ in E_i, and since E_i and $\mathscr{N}(E_i)$ by Definition 17 have the same alphabet, it holds that $x_i \xrightarrow{\sigma:\Delta_{\mathscr{N}(\mathscr{E})}(\sigma)} y_i$ in $\mathscr{N}(E_i)$. If $\sigma \notin \Sigma_i$, then σ is not in the alphabet of $\mathscr{N}(E_i)$ and $x_i = y_i$. Then $(x_1, \ldots, x_n) \xrightarrow{\sigma:\Delta_{\mathscr{N}(\mathscr{E})}(\sigma)} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{N}(\mathscr{E})$ by Definition 14, and as $\Delta_{\mathscr{N}(\mathscr{E})}(\sigma) \equiv p$, it holds that $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{N}(\mathscr{E})$.

Conversely, assume $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{N}(\mathscr{E})$ where $p \equiv \Delta_{\mathscr{N}(\mathscr{E})}(\sigma) \equiv \bigwedge_{i:\sigma \in \Sigma_i} \Delta_i(\sigma)$. Consider two cases for each E_i: either $\sigma \in \Sigma_i$ or $\sigma \notin \Sigma_i$. If $\sigma \in \Sigma_i$, then by Definition 14 it follows that $x_i \xrightarrow{\sigma:p} y_i$ in $\mathscr{N}(E_i)$, and since E_i and $\mathscr{N}(E_i)$ have the same alphabet, it holds that $x_i \xrightarrow{\sigma:\Delta_i(\sigma)} y_i$ in E_i. If $\sigma \notin \Sigma_i$, then σ is not in the alphabet of E_i and $x_i = y_i$. By Definition 14, it follows that $(x_1, \ldots, x_n) \xrightarrow{\sigma:\bigwedge_{i:\sigma\in\Sigma_i} \Delta_i(\sigma)} (y_1, \ldots, y_n)$ in $||\mathscr{E}$, and as $\bigwedge_{i:\sigma\in\Sigma_i} \Delta_i(\sigma) \equiv p$, it holds that $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $||\mathscr{E}$. □

B EFSM-based compositional verification

This appendix contains the proofs of the results concerning abstraction methods presented in Section 5. Each of the following subsections contains the proofs for the propositions in the corresponding subsection of Section 5.

B.1 FSM-based conflict equivalence abstraction

This section contains the proof of Proposition 5 in Section 5.1, which states that the nonblocking property of an EFSM system is preserved when the FSM form of a single component is simplified subject to conflict equivalence of FSMs. This proof requires modular reasoning about the unfolded EFSM system to exploit the conflict equivalence of FSM forms.

This reasoning is facilitated using an alternative way to unfold an EFSM system, called *modular unfolding*, where the variables are unfolded to a single *variable FSM* while the EFSMs are only replaced by their FSM forms.

Definition 27 Let $\mathscr{E}$ be a normalised EFSM system with variable set $V = \mathrm{vars}(\mathscr{E})$. The *variable FSM* for $\mathscr{E}$ is $V_{\mathscr{E}} = \langle \Sigma_{\mathscr{E}}, \mathrm{dom}(V), \rightarrow_V, \{v^\circ\}, \mathrm{dom}(V)\rangle$ where $\hat{v} \overset{\sigma}{\rightarrow}_V \hat{w}$ if $\Xi_V(\Delta_{\mathscr{E}}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$.

Definition 28 Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system. The *modular unfolding* of $\mathscr{E}$ is

$$\varphi(E_1)|| \cdots ||\varphi(E_n)||V_{\mathscr{E}} \ . \tag{26}$$

The variable FSM $V_{\mathscr{E}}$ has all possible valuations of the variables of $\mathscr{E}$ as its states and in its transitions encodes all the constraints imposed by the updates. This makes it possible to replace each EFSM E_i by its FSM form $\varphi(E_i)$ according to Definition 19, resulting in the system (26) of FSMs that interact in standard FSM synchronous composition (Definition 2). The following Lemma 13 shows that this modular unfolding is isomorphic to the EFSM system unfolding $U(\mathscr{E})$. Then the modular unfolding can be used to decompose an EFSM system into FSMs and prove Proposition 5.

Lemma 13 *Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system. Then $U(\mathscr{E}) = \varphi(E_1)|| \cdots ||\varphi(E_n)||V_{\mathscr{E}}$.*

Proof Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ with $E_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q^\circ_\omega, Q^\omega_i\rangle$, let $E = U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$ by Proposition 1, and let $F = \varphi(E_1)|| \cdots ||\varphi(E_n)||V_{\mathscr{E}}$. Since E_i and $\varphi(E_i)$ have the same alphabet Σ_i, it follows that $\varphi(E_1)|| \cdots ||\varphi(E_n)$ and $\mathscr{E}$ also have the same alphabet $\Sigma_{\mathscr{E}} = \bigcup_{i=1}^n \Sigma_i$. The alphabet of $V_{\mathscr{E}}$ also is $\Sigma_{\mathscr{E}}$, which implies that $\Sigma_E = \Sigma_{\mathscr{E}} = \Sigma_F$. Moreover, by Definition 27 it holds that $Q_E = Q_1 \times \ldots \times Q_n \times \mathrm{dom}(V) = Q_F$, $Q{\circ}_E = Q\circ_1 \times \ldots \times Q\circ_n \times \{\hat{v}\circ\} = Q{\circ}_F$, and $Q^\omega_E = Q^\omega_1 \times \ldots \times Q^\omega_n \times \mathrm{dom}(V) = Q^\omega_F$. It is left to show that $\rightarrow_E = \rightarrow_F$.

First let $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $E = U(\dot{\|}\mathscr{E})$. This means by Definition 10 that $(x_1, \ldots, x_n) \xrightarrow{\sigma : p} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{E}$ where $p \equiv \Delta_{\mathscr{E}}(\sigma)$ and $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$. The latter means by Definition 27 that $\hat{v} \xrightarrow{\sigma}_V \hat{w}$ in $V_{\mathscr{E}}$. Now consider two cases for each E_i: either $\sigma \in \Sigma_i$ or $\sigma \notin \Sigma_i$. If $\sigma \in \Sigma_i$, it follows by Definition 14 that $x_i \xrightarrow{\sigma : p} y_i$ in E_i, which implies $x_i \xrightarrow{\sigma} y_i$ in $\varphi(E_i)$. If $\sigma \notin \Sigma_i$ then $x_i = y_i$. Thus, $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $\varphi(E_1) \| \cdots \| \varphi(E_n) \| V_{\mathscr{E}} = F$.

Conversely, let $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $F = \varphi(E_1) \| \cdots \| \varphi(E_n) \| V_{\mathscr{E}}$. This means $(x_1, \ldots, x_n) \xrightarrow{\sigma} (y_1, \ldots, y_n)$ in $\varphi(E_1) \| \ldots \| \varphi(E_n)$ and $\hat{v} \xrightarrow{\sigma} \hat{w}$ in $V_{\mathscr{E}}$. Consider two cases for each E_i: either $\sigma \in \Sigma_i$ or $\sigma \notin \Sigma_i$. If $\sigma \in \Sigma_i$ then by Definition 2 it follows that $x_i \xrightarrow{\sigma} y_i$ in $\varphi(E_i)$. Then by Definition 19 it holds that $x_i \xrightarrow{\sigma : p} y_i$ in E_i, where $p \equiv \Delta_{E_i}(\sigma) \equiv \Delta_{\mathscr{E}}(\sigma)$ as $\mathscr{E}$ is normalised. If $\sigma \notin \Sigma_i$ then $x_i = y_i$. Thus, $(x_1, \ldots, x_n) \xrightarrow{\sigma : p} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{E}$ by Definition 14. Furthermore, as $\hat{v} \xrightarrow{\sigma} \hat{w}$ in $V_{\mathscr{E}}$, it holds by Definition 27 that $\Xi_V(p)(\hat{v}, \hat{w}) = \Xi_V(\Delta_{\mathscr{E}}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$. It follows by Definition 10 that $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\dot{\|}\mathscr{E}) = E$. □

Proposition 5 *Let $\mathscr{E} = \{E_1, E_2, \ldots, E_n\}$ be a normalised EFSM system and let $\Upsilon \subseteq \Sigma_1$ such that $(\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset$ and $\Delta_{\mathscr{E}}(\sigma) \equiv true$ for all $\sigma \in \Upsilon$. Let $\mathscr{F} = \{F_1, E_2, \ldots, E_n\}$ be a normalised EFSM system such that $\varphi(E_1) \setminus \Upsilon \simeq_{conf} \varphi(F_1) \setminus \Upsilon$. Then $\mathscr{E}$ is nonblocking if and only if $\mathscr{F}$ is nonblocking.*

Proof Let

$$\varphi(E_1) \setminus \Upsilon \simeq_{conf} \varphi(F_1) \setminus \Upsilon \,. \tag{27}$$

Because of symmetry it is enough to show that, if $\mathscr{E}$ is nonblocking then $\mathscr{F}$ is nonblocking. Therefore assume that $\mathscr{E}$ is nonblocking, which means that $U(\mathscr{E})$ is nonblocking. By Lemma 13,

$$U(\mathscr{E}) = \varphi(E_1) \| \cdots \| \varphi(E_n) \| V_{\mathscr{E}} \tag{28}$$

is nonblocking. As $\Delta_{\mathscr{E}}(\upsilon) \equiv true$ for all $\upsilon \in \Upsilon$, it holds by Definition 27 that $\hat{v} \xrightarrow{\upsilon} \hat{v}$ in $V_{\mathscr{E}}$ for all $\hat{v} \in \mathrm{dom}(\mathrm{vars}(\mathscr{E}))$ and all $\upsilon \in \Upsilon$, and these events appear on no other transitions in $V_{\mathscr{E}}$. These events are pure selfloop events in $V_{\mathscr{E}}$ and can be removed (Wonham), i.e.,

$$U(\mathscr{E}) = \varphi(E_1) \| \cdots \| \varphi(E_n) \| V_{\mathscr{E}} = \varphi(E_1) \| \cdots \| \varphi(E_n) \| V_{\mathscr{E}|\Omega} \tag{29}$$

is nonblocking, where $\Omega = \Sigma_{\mathscr{E}} \setminus \Upsilon$. Now consider $T = \varphi(E_2) \| \ldots \| \varphi(E_n) \| V_{\mathscr{E}|\Omega}$. Then it follows from $(\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset$ and $\Omega \cap \Upsilon = \emptyset$ that

$$(\varphi(E_1) \setminus \Upsilon) \| \varphi(E_2) \| \ldots \| \varphi(E_n) \| V_{\mathscr{E}|\Omega} \tag{30}$$

is nonblocking. Note that $V_{\mathscr{E}} = V_{\mathscr{F}}$. Since $\varphi(E_1) \setminus \Upsilon$ and $\varphi(F_1) \setminus \Upsilon$ are conflict equivalent (27), it follows that

$$(\varphi(F_1) \setminus \Upsilon) \| \varphi(E_2) \| \ldots \| \varphi(E_n) \| V_{\mathscr{F}|\Omega} \tag{31}$$

is nonblocking. Again since $(\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset$ and $\Omega \cap \Upsilon = \emptyset$ and the events in Υ are pure selfloops in $V_{\mathscr{E}} = V_{\mathscr{F}}$, it follows that

$$\varphi(F_1) \| \varphi(E_2) \| \ldots \| \varphi(E_n) \| V_{\mathscr{F}|\Omega} = \varphi(F_1) \| \varphi(E_2) \| \ldots \| \varphi(E_n) \| V_{\mathscr{F}} = U(\mathscr{F}) \tag{32}$$

is nonblocking, i.e., $\mathscr{F}$ is nonblocking. □

B.2 Partial composition

This section proves that the synchronous composition of two components in an EFSM system preserves the nonblocking property of the system as stated in Proposition 6 in Section 5.2. In this proof, it is shown that the results of unfolding before and after partial synchronous composition are not only equivalent but identical.

Proposition 6 *(Partial Composition) Let* $\mathscr{E} = \{E_1, \ldots, E_n\}$ *be an EFSM system, and let* $\mathscr{F} = \{E_1 \dot{\|} E_2, E_3, \ldots, E_n\}$*. Then* $\dot{\|}\mathscr{E} = \dot{\|}\mathscr{F}$*.*

Proof It follows from Definition 14 that $\dot{\|}\mathscr{E}$ and $\dot{\|}\mathscr{F}$ have the same event and state sets, including initial and marked states. It remains to be shown that they also have the same transitions. Throughout the proof, let $\Sigma_i = \Sigma_{E_i}$ for $1 \le i \le n$.

First, let

$$(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n) \tag{33}$$

in $\dot{\|}\mathscr{E}$. By Definition 14, this means for each $1 \le i \le n$ that either $\sigma \in \Sigma_i$ and $x_i \xrightarrow{\sigma:p} y_i$ or $\sigma \notin \Sigma_i$ and $x_i = y_i$. Consider four cases for E_1 and E_2.

- If $\sigma \in \Sigma_1 \cap \Sigma_2$, then $x_1 \xrightarrow{\sigma:p} y_1$ in E_1 and $x_2 \xrightarrow{\sigma:p} y_2$ in E_2, and by Definition 14 it holds that $(x_1, x_2) \xrightarrow{\sigma:p} (y_1, y_2)$ in $E_1 \dot{\|} E_2$.
- If $\sigma \in \Sigma_1 \setminus \Sigma_2$, then $x_1 \xrightarrow{\sigma:p} y_1$ in E_1 and $x_2 = y_2$, and by Definition 14 it holds that $(x_1, x_2) \xrightarrow{\sigma:p} (y_1, x_2) = (y_1, y_2)$ in $E_1 \dot{\|} E_2$.
- If $\sigma \in \Sigma_2 \setminus \Sigma_1$, then $x_1 = y_1$ and $x_2 \xrightarrow{\sigma:p} y_2$ in E_2, and by Definition 14 it holds that $(x_1, x_2) \xrightarrow{\sigma:p} (x_1, y_2) = (y_1, y_2)$ in $E_1 \dot{\|} E_2$.
- If $\sigma \notin \Sigma_1 \cup \Sigma_2$, then σ is not in the alphabet of $E_1 \dot{\|} E_2$ and $(x_1, x_2) = (y_1, y_2)$.

Combining the above observations for $E_1 \dot{\|} E_2$ and $E_3, \ldots, E_n$, it follows by Definition 14 that $((x_1, x_2), x_3, \ldots, x_n) \xrightarrow{\sigma:p} ((y_1, y_2), y_3, \ldots, y_n)$ in $\dot{\|}\mathscr{F}$.

Conversely, let

$$((x_1, x_2), x_3, \ldots, x_n) \xrightarrow{\sigma:p} ((y_1, y_2), y_3, \ldots, y_n) \tag{34}$$

in $\dot{\|}\mathscr{F}$. Consider four cases for E_1 and E_2.

- If $\sigma \in \Sigma_1 \cap \Sigma_2$, then by Definition 14 it holds that $(x_1, x_2) \xrightarrow{\sigma:p} (y_1, y_2)$ in $E_1 \dot{\|} E_2$, and furthermore $x_1 \xrightarrow{\sigma:p} y_1$ in E_1 and $x_2 \xrightarrow{\sigma:p} y_2$ in E_2.
- If $\sigma \in \Sigma_1 \setminus \Sigma_2$, then by Definition 14 it holds that $(x_1, x_2) \xrightarrow{\sigma:p} (y_1, y_2)$ in $E_1 \dot{\|} E_2$, and furthermore $x_1 \xrightarrow{\sigma:p} y_1$ in E_1 and $\sigma \notin \Sigma_2$ and $x_2 = y_2$.
- If $\sigma \in \Sigma_2 \setminus \Sigma_1$, then by Definition 14 it holds that $(x_1, x_2) \xrightarrow{\sigma:p} (y_1, y_2)$ in $E_1 \dot{\|} E_2$, and furthermore $\sigma \notin \Sigma_1$ and $x_1 = y_1$ and $x_2 \xrightarrow{\sigma:p} y_2$ in E_2 and
- if $\sigma \notin \Sigma_1 \cup \Sigma_2$, then by Definition 14 it holds that $x_1 = x_2$ and $y_1 = y_2$.

Furthermore, for $3 \le i \le n$ it follows from (34) by Definition 14 that either $\sigma \in \Sigma_i$ and

$x_i \xrightarrow{\sigma:p} y_i$ or $\sigma \notin \Sigma_i$ and $x_i = y_i$. Combining the above observations for $E_1, \ldots, E_n$, it follows by Definition 14 that $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{F}$. □

B.3 Update simplification

This section proves the correctness of update simplification as stated in Proposition 7 in Section 5.3. The proof uses the following lemma, which shows two EFSM systems with logically equivalent updates with respect to all variables have isomorphic monolithic flattening results.

Lemma 14 *Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ and $\mathscr{F} = \{F_1, \ldots, F_n\}$ be normalised EFSM systems with $E_i = \langle \Sigma_i, Q_i, \rightarrow_i^E, Q_i^\circ, Q_i^\omega \rangle$ and $F_i = \langle \Sigma_i, Q_i, \rightarrow_i^F, Q_i^\circ, Q_i^\omega \rangle$. Let $V = vars(\mathscr{E}) = vars(\mathscr{F})$ and $\Delta_\mathscr{E}(\sigma) \Leftrightarrow_V \Delta_\mathscr{F}(\sigma)$ for all $\sigma \in \Sigma_\mathscr{E} = \Sigma_\mathscr{F}$, and $\rightarrow_i^F = \{\, (x, \sigma, \Delta_\mathscr{F}(\sigma), y) \mid x \xrightarrow{\sigma:\Delta_\mathscr{E}(\sigma)}{}_i^E\, y \,\}$. Then $U(\mathscr{E}) = U(\mathscr{F})$.*

Proof Clearly, $U(\mathscr{E})$ and $U(\mathscr{F})$ by construction both have the same event alphabet $\Sigma_\mathscr{E}$, and they have the same state sets, including initial and marked states. Also note that $\mathscr{E}$ and $\mathscr{F}$ are normalised, so $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$ and $U(\mathscr{F}) = U(\dot{\|}\mathscr{F})$ by Proposition 1. It remains to be shown that $U(\mathscr{E})$ and $U(\mathscr{F})$ have the same transitions. Because of symmetry it is enough to show that, if $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{E})$ then $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{F})$.

Assume $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$. By Definition 10, this means $(x_1, \ldots, x_n) \xrightarrow{\sigma:\Delta_\mathscr{E}(\sigma)} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{E}$ and $\Xi_V(\Delta_\mathscr{E}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$. Then by construction, $(x_1, \ldots, x_n) \xrightarrow{\sigma:\Delta_\mathscr{F}(\sigma)} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{F}$ and $\Delta_\mathscr{E}(\sigma) \Leftrightarrow_V \Delta_\mathscr{F}(\sigma)$. The latter means $\Xi_V(\Delta_\mathscr{E}(\sigma)) \Leftrightarrow \Xi_V(\Delta_\mathscr{F}(\sigma))$ by Definition 20, i.e., $\Xi_V(\Delta_\mathscr{F}(\sigma))(\hat{v}, \hat{w}) = \Xi_V(\Delta_\mathscr{E}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$. It follows by Definition 10 that $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\dot{\|}\mathscr{F}) = U(\mathscr{F})$. □

Proposition 7 (Update Simplification) *Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ and $\mathscr{F} = \{F_1, \ldots, F_n\}$ be normalised EFSM systems with $E_i = \langle \Sigma_i, Q_i, \rightarrow_i^E, Q_i^\circ, Q^\omega[i] \rangle$ and $F_i = \langle \Sigma_i, Q_i, \rightarrow_i^F, Q_i^\circ, Q_i^\omega \rangle$. Let $V = vars(\mathscr{E}) = vars(\mathscr{F})$ and $\Delta_\mathscr{E}(\sigma) \Leftrightarrow_V \Delta_\mathscr{F}(\sigma)$ for all $\sigma \in \Sigma_\mathscr{E} = \Sigma_\mathscr{F}$, and $\rightarrow_i^F = \{\, (x, \sigma, \Delta_\mathscr{F}(\sigma), y) \mid x \xrightarrow{\sigma:\Delta_\mathscr{E}(\sigma)}{}_i^E\, y \,\}$. Then $\mathscr{E}$ is nonblocking if and only if $\mathscr{F}$ is nonblocking.*

Proof By Definition 11, $\mathscr{E}$ is nonblocking if and only if $U(\mathscr{E})$ is nonblocking, and $\mathscr{F}$ is nonblocking if and only if $U(\mathscr{F})$ is nonblocking; and by Lemma 14, it holds that $U(\mathscr{E}) = U(\mathscr{F})$. It follows that $\mathscr{E}$ is nonblocking if and only if $\mathscr{F}$ is nonblocking. □

B.4 Variable unfolding

This section proves that unfolding of a variable in an EFSM system preserves the nonblocking property of the system as stated in Proposition 8 in Section 5.4. The key step to prove this result is contained in Lemma 15, which shows that the FSMs obtained from completely unfolding the system before and after partial unfolding have essentially the same transition relations. The link between these transition relations is established by extending or restricting valuations to add or remove the variable to be unfolded. The following two definitions are needed for this purpose.

Definition 29 Let $\hat{v}\colon V \to D$ be a valuation. For a variable set $W \subseteq V$, the *restriction* $\hat{v}_{|W}\colon W \to D$ is defined by $\hat{v}_{|W}[v] = \hat{v}[v]$ for all $v \in W$.

Definition 30 Let $V = V_1 \dot{\cup} V_2$ be a variable set, and let $\hat{v}_1\colon V_1 \to D_1$ and $\hat{v}_2\colon V_2 \to D_2$ be two valuations. The *extension* $\hat{v}_1 \oplus \hat{v}_2\colon V \to D_1 \cup D_2$ is defined by

$$(\hat{v}_1 \oplus \hat{v}_2)[v] = \begin{cases} \hat{v}_1[v], & \text{if } v \in V_1 \ ; \\ \hat{v}_2[v], & \text{if } v \in V_2 \ . \end{cases} \tag{35}$$

Lemma 15 *Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system and $z \in vars(\mathcal{E})$. Then $(a, x_1, \ldots, x_n, \check{v}) \xrightarrow{\sigma} (b, x_1, \ldots, x_n, \check{w})$ in $\rho_z(U(\mathcal{E} \backslash z))$ if and only if $(x_1, \ldots, x_n, \check{v} \oplus \{z \mapsto a\}) \xrightarrow{\sigma} (x_1, \ldots, x_n, \check{w} \oplus \{z \mapsto b\})$ in $U(\mathcal{E})$.*

Proof Let $V = \mathrm{vars}(\mathcal{E})$, and let $\hat{v} = \check{v} \oplus \{z \mapsto a\}$ and $\hat{w} = \check{w} \oplus \{z \mapsto b\}$, which means $\hat{v}[z] = a$ and $\hat{w}[z] = b$. Also write $E_i = \langle \Sigma_i, Q_i, \to_i, Q_i^{\circ}, Q_i^{\omega} \rangle$ for $1 \le i \le n$. Note that $\mathcal{E}$ and $\mathcal{E} \setminus z$ are normalised, so by Proposition 1 it holds that $U(\mathcal{E}) = U(\dot{\|}\mathcal{E})$ and $U(\mathcal{E} \setminus z) = U(\dot{\|}(\mathcal{E} \setminus z))$.

First let $(a, x_1, \ldots, x_n, \check{v}) \xrightarrow{\sigma} (b, y_1, \ldots, y_n, \check{w})$ in $\rho_z(U(\mathcal{E} \setminus z)) = \rho_z(U(\dot{\|}\mathcal{E} \setminus z))$. Note that $\mathcal{E} \setminus z = \{U_{\mathcal{E}}(z), U_z(E_1), \cdots, U_z(E_n)\}$ by Definition 24. Consider two cases.

(i) $\sigma \in \Sigma_z$. Then $z \in \mathrm{vars}(\Delta_{\mathcal{E}}(\sigma))$ by Definition 22, and $(a, x_1, \ldots, x_n, \check{v}) \xrightarrow{(\sigma', a', b')} (b, y_1, \ldots, y_n, \check{w})$ in $U(\mathcal{E} \setminus z) = U(\dot{\|}(\mathcal{E} \setminus z))$ for some $(\sigma', a', b') \in U_z(\Sigma_z)$ such that $\rho_z((\sigma', a', b')) = \sigma$. By definition of ρ_z it holds that $\sigma' = \sigma$. By Definition 10, it holds that

$$\begin{aligned} &(a, x_1, \ldots, x_n) \xrightarrow{(\sigma, a', b') : \Delta_{\mathcal{E}\backslash z}((\sigma, a', b'))} (b, y_1, \ldots, y_n) \quad \text{in } \dot{\|}(\mathcal{E} \setminus z) \\ &= U_{\mathcal{E}}(z) \dot{\|} U_z(E_1) \dot{\|} \cdots \dot{\|} U_z(E_n) \end{aligned} \tag{36}$$

and $\Xi_{V\backslash\{z\}}(\Delta_{\mathcal{E}\backslash z}((\sigma, a', b')))(\check{v}, \check{w}) = \mathbf{T}$. As $(\sigma, a', b') = (\sigma', a', b') \in U_z(\Sigma_z)$ is in the alphabet of $U_{\mathcal{E}}(z)$, it follows that

$$a \xrightarrow{(\sigma, a', b') : \Delta_{\mathcal{E}\backslash z}((\sigma, a', b'))} b \quad \text{in } U_{\mathcal{E}}(z) \ . \tag{37}$$

Then it follows from Definition 22 that $a' = a$ and $b' = b$, and $\Delta_{\mathcal{E}\backslash z}((\sigma, a, b)) \equiv \Xi_{\{z\}}(\Delta_{\mathcal{E}}(\sigma))[z \mapsto a, z' \mapsto b]$. Note that

$$\begin{aligned} \Xi_V(\Delta_{\mathcal{E}}(\sigma))(\hat{v}, \hat{w}) &= \Xi_V(\Delta_{\mathcal{E}}(\sigma))(\check{v} \oplus \{z \mapsto a\}, \check{w} \oplus \{z \mapsto b\}) \\ &= \Xi_{V\backslash\{z\}}(\Xi_{\{z\}}(\Delta_{\mathcal{E}}(\sigma)))(\check{v} \oplus \{z \mapsto a\}, \check{w} \oplus \{z \mapsto b\}) \\ &= \Xi_{V\backslash\{z\}}(\Xi_{\{z\}}(\Delta_{\mathcal{E}}(\sigma)))[z \mapsto a, z' \mapsto b](\check{v}, \check{w}) \\ &= \Xi_{V\backslash\{z\}}(\Xi_{\{z\}}(\Delta_{\mathcal{E}}(\sigma))[z \mapsto a, z' \mapsto b])(\check{v}, \check{w}) \\ &= \Xi_{V\backslash\{z\}}(\Delta_{\mathcal{E}\backslash z}((\sigma, a, b)))(\check{v}, \check{w}) \\ &= \Xi_{V\backslash\{z\}}(\Delta_{\mathcal{E}\backslash z}((\sigma, a', b')))(\check{v}, \check{w}) \\ &= \mathbf{T} \ . \end{aligned}$$

Now consider some E_i with $1 \le i \le n$. If $\sigma \in \Sigma_i$ then since $\sigma \in \Sigma_z$ also $(\sigma, a', b') \in U_z(\Sigma_i)$ so that (σ, a', b') is in the alphabet of $U_z(E_i)$ by Definition 23. It follows from (36) that $x_i \xrightarrow{(\sigma,a',b'):\Delta_{\mathscr{E}\backslash z}((\sigma,a',b'))} y_i$ in $U_z(E_i)$, which implies $x_i \xrightarrow{\sigma:\Delta_{\mathscr{E}}(\sigma)} y_i$ in E_i by Definition 23. Otherwise, if $\sigma \notin \Sigma_i$ then (σ, a', b') is not in the alphabet of $U_z(E_i)$ and $x_i = y_i$. Having shown the above for all $1 \le i \le n$, it can be concluded by Definition 14 that $(x_1, \ldots, x_n) \xrightarrow{\sigma:\Delta_{\mathscr{E}}(\sigma)} (y_1, \ldots, y_n)$ in $E_1 \dot{\|} \cdots \dot{\|} E_n = \dot{\|}\mathscr{E}$.

(ii) $\sigma \notin \Sigma_z$. Then $z \notin \mathrm{vars}(\Delta_{\mathscr{E}}(\sigma))$ by Definition 22, and $(a, x_1, \ldots, x_n, \check{v}) \xrightarrow{\sigma'} (b, y_1, \ldots, y_n, \check{w})$ in $U(\mathscr{E} \setminus z) = U(\dot{\|}(\mathscr{E} \setminus z))$ for some $\sigma' \in \Sigma_{\mathscr{E}\backslash z} \setminus U_z(\Sigma_z)$ such that $\rho_z(\sigma') = \sigma$. By definition of ρ_z it holds that $\sigma' = \sigma \in \Sigma_{\mathscr{E}}$. By Definition 10, it holds that

$$(a, x_1, \ldots, x_n) \xrightarrow{\sigma:\Delta_{\mathscr{E}\backslash z}(\sigma)} (b, y_1, \ldots, y_n) \quad \text{in } \dot{\|}(\mathscr{E}\backslash z) = U_{\mathscr{E}}(z) \dot{\|} U_z(E_1) \dot{\|} \cdots \dot{\|} U_z(E_n) \tag{38}$$

and $\Xi_{V\backslash\{z\}}(\Delta_{\mathscr{E}\backslash z}(\sigma))(\check{v}, \check{w}) = \mathbf{T}$. As $\sigma \in \Sigma_{\mathscr{E}}$, it is clear that $\sigma \notin U_z(\Sigma_z)$ and thus σ is not in the alphabet of $U_{\mathscr{E}}(z)$, which implies $a = b$. Also by (38), there must exist i such that $x_i \xrightarrow{\sigma:\Delta_{\mathscr{E}\backslash z}(\sigma)} y_i$ in $U_z(E_i)$, which given $\sigma \in \Sigma_{\mathscr{E}}$ implies $x_i \xrightarrow{\sigma:\Delta_{\mathscr{E}\backslash z}(\sigma)} y_i$ in E_i by Definition 23 where $\Delta_{\mathscr{E}\backslash z}(\sigma) \equiv \Delta_{\mathscr{E}}(\sigma)$ as $\mathscr{E}$ is normalised. As $z \notin \mathrm{vars}(\Delta_{\mathscr{E}}(\sigma))$, it holds that $\Xi_V(\Delta_{\mathscr{E}}(\sigma))(\hat{v}, \hat{w}) = \Xi_V(\Delta_{\mathscr{E}}(\sigma))(\check{v} \oplus \{z \mapsto a\}, \check{w} \oplus \{z \mapsto b\}) = \Xi_{V\backslash\{z\}}(\Delta_{\mathscr{E}}(\sigma))(\check{v}, \check{w}) = \Xi_{V\backslash\{z\}}(\Delta_{\mathscr{E}\backslash z}(\sigma))(\check{v}, \check{w}) = \mathbf{T}$.

Now consider some E_i with $1 \le i \le n$. If $\sigma \in \Sigma_i$ then since $\sigma \notin \Sigma_z$ it follows from (38) that $x_i \xrightarrow{\sigma:\Delta_{\mathscr{E}\backslash z}(\sigma)} y_i$ in $U_z(E_i)$, which implies $x_i \xrightarrow{\sigma:\Delta_{\mathscr{E}}(\sigma)} y_i$ in E_i by Definition 23. Otherwise, if $\sigma \notin \Sigma_i$ then σ is not in the alphabet of $U_z(E_i)$ and $x_i = y_i$. Having shown the above for all $1 \le i \le n$, it can be concluded by Definition 14 that $(x_1, \ldots, x_n) \xrightarrow{\sigma:\Delta_{\mathscr{E}}(\sigma)} (y_1, \ldots, y_n)$ in $E_1 \dot{\|} \cdots \dot{\|} E_n = \dot{\|}\mathscr{E}$.

In both cases, it has been shown that $(x_1, \ldots, x_n) \xrightarrow{\sigma:\Delta_{\mathscr{E}}(\sigma)} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{E}$ and $\Xi_V(\Delta_{\mathscr{E}}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$. Then it follows by Definition 10 that $(x_1, \ldots, x_n, \check{v} \oplus \{z \mapsto a\}) = (x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w}) = (y_1, \ldots, y_n, \check{w} \oplus \{z \mapsto b\})$ in $U(\dot{\|}\mathscr{E}) = U(\mathscr{E})$.

Conversely let $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$. Then it holds by Definition 10 that

$$(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n) \qquad \text{in } \dot{\|}\mathscr{E} = E_1 \dot{\|} \cdots \dot{\|} E_n \tag{39}$$

where $p \equiv \Delta_{\mathscr{E}}(\sigma)$ and $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$. Consider two cases.

(i) $\sigma \in \Sigma_z$. Note that $z \in \mathrm{vars}(\Delta_{\mathscr{E}}(\sigma)) = \mathrm{vars}(p)$. Then by Definition 22 it holds that $a \xrightarrow{(\sigma,a,b):p'} b$ in $U_{\mathscr{E}}(z)$ where $p' \equiv \Xi_{\{z\}}(p)[z \mapsto a, z' \mapsto b]$ and $\rho_z((\sigma, a, b)) = \sigma$. Note that $\Xi_{V\backslash\{z\}}(p')(\check{v}, \check{w}) = \Xi_{V\backslash\{z\}}(\Xi_{\{z\}}(p))[z \mapsto a, z' \mapsto b])(\check{v}, \check{w}) = (\Xi_V(p)[z \mapsto a, z' \mapsto b])(\check{v}, \check{w}) = \Xi_V(p)(\check{v} \oplus \{z \mapsto b\}, \check{w} \oplus \{z \mapsto b\}) = \Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$.

Now consider some E_i with $1 \le i \le n$. If $\sigma \in \Sigma_i$, it follows from (39) that $x_i \xrightarrow{\sigma:p} y_i$ in E_i, which implies $x_i \xrightarrow{(\sigma,a,b):p'} y_i$ in $U_z(E_i)$ by Definition 23 as $\sigma \in \Sigma_z$. Otherwise, if $\sigma \notin \Sigma_i$ then σ is not in the alphabet of E_i and $x_i = y_i$. Having shown the above for all $1 \le i \le n$, it can be concluded by Definition 14 that

$$(a, x_1, \ldots, x_n) \xrightarrow{(\sigma,a,b):p'} (b, y_1, \ldots, y_n) \text{ in } U_{\mathscr{E}}(z) \dot{\|} U_z(E_1) \dot{\|} \cdots ||U_z(E_n) = \dot{\|}(\mathscr{E} \setminus z)\,. \tag{40}$$

Since $\Xi_{V\setminus\{z\}}(p')(\check{v}, \check{w}) = \mathbf{T}$, it follows by Definition 10 that $(a, x_1, \ldots, x_n, \check{v}) \xrightarrow{(\sigma,a,b):p'} (b, y_1, \ldots, y_n, \check{w})$ in $U(\dot{\|}(\mathcal{E} \setminus z)) = U(\mathcal{E} \setminus z)$, which implies $(a, x_1, \ldots, x_n, \check{v}) \xrightarrow{\sigma} (b, y_1, \ldots, y_n, \check{w})$ in $\rho_z(U(\mathcal{E} \setminus z))$.

(ii) $\sigma \notin \Sigma_z$. In this case, by Definition 22 it holds that $z \notin \mathrm{vars}(\Delta_{\mathcal{E}}(\sigma)) = \mathrm{vars}(p)$ and $\rho_z(\sigma) = \sigma \in \Sigma_{\mathcal{E}}$ is not in the alphabet of $U_{\mathcal{E}}(z)$. Consider some E_i with $1 \le i \le n$. If $\sigma \in \Sigma_i$, it follows from (39) that $x_i \xrightarrow{\sigma:p} y_i$ in E_i, which implies $x_i \xrightarrow{\sigma:p} y_i$ in $U_z(E_i)$ by Definition 23 as $\sigma \in \Sigma_i \setminus \Sigma_z$. Otherwise, if $\sigma \notin \Sigma_i$ then σ is not in the alphabet of E_i and $x_i = y_i$. Having shown the above for all $1 \le i \le n$, it can be concluded by Definition 14 that

$$(a, x_1, \ldots, x_n) \xrightarrow{\sigma:p} (a, y_1, \ldots, y_n) \quad \text{in } U_{\mathcal{E}}(z) \dot{\|} U_z(E_1) \dot{\|} \cdots \dot{\|} U_z(E_n) = \dot{\|}(\mathcal{E} \setminus z)\,. \tag{41}$$

From $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$ and $z \notin \mathrm{vars}(p) \supseteq \mathrm{vars}'(p)$, it follows that $(z' = z)(\hat{v}, \hat{w}) = \mathbf{T}$. This means $a = \hat{v}[z] = \hat{w}[z] = b$ and $\Xi_{V\setminus\{z\}}(p)(\check{v}, \check{w}) = \Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$. Then by Definition 10, it holds that $(a, x_1, \ldots, x_n, \check{v}) \xrightarrow{\sigma} (a, y_1, \ldots, y_n, \check{w}) = (b, y_1, \ldots, y_n, \check{w})$ in $U(\dot{\|}(\mathcal{E} \setminus z)) = U(\mathcal{E} \setminus z)$, which given $\rho_z(\sigma) = \sigma$ implies $(a, x_1, \ldots, x_n, \check{v}) \xrightarrow{\sigma} (b, y_1, \ldots, y_n, \check{w})$ in $\rho_z(U(\mathcal{E} \setminus z))$.

□

Proposition 8 (Variable Unfolding) *Let $\mathcal{E}$ be a normalised EFSM system, and let $z \in \mathrm{vars}(\mathcal{E})$. Then $\mathcal{E}$ is nonblocking if and only if $\mathcal{E} \setminus z$ is nonblocking.*

Proof Let $\mathcal{E} = \{E_1, \ldots, E_n\}$, let $\mathcal{E} \setminus z = \{U_{\mathcal{E}}(z), U_z(E_1), \ldots, U_z(E_n)$, according to Definition 24, and let $\rho_z \colon \Sigma_{\mathcal{E}} \cup U_z(\Sigma_z) \to \Sigma_{\mathcal{E}}$ be the variable renaming map according to Definition 22.

First assume $\mathcal{E}$ is nonblocking, which implies $U(\mathcal{E})$ is nonblocking. It will be shown that $\rho_z(U(\mathcal{E} \setminus z))$ is nonblocking. Assume $(a^0, x_1^0, \ldots, x_n^0, \check{v}^0) \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_m} (a^l, x_1^l, \ldots, x_n^l, \check{v}^l)$ in $\rho_z(U(\mathcal{E} \setminus z))$. From Lemma 15 it follows that $(x_1^0, \ldots, x_n^0, \check{v}^0 \oplus \{z \mapsto a^0\}) \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_m} (x_1^l, \ldots, x_n^l, \check{v}^l \oplus \{z \mapsto a^l\})$ in $U(\mathcal{E})$. Since $U(\mathcal{E})$ is nonblocking, there exists a path $(x_1^l, \ldots, x_n^l, \hat{v}^l) \xrightarrow{\sigma_{l+1}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \hat{v}^m)$ in $U(\mathcal{E})$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. From Lemma 15 it follows that $(a^l, x_1^l, \ldots, x_n^l, \check{v}^l) \xrightarrow{\sigma_{l+1}} \cdots \xrightarrow{\sigma_l} (a^m, x_1^m, \ldots, x_n^m, \check{v}^m)$ in $\rho_z(U(\mathcal{E} \setminus z))$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$ and $\hat{v}^i = \check{v}^i \oplus \{z \mapsto a^i\}$ for $l + 1 \le i \le m$. Since $(x_1^l, \ldots, x_n^l, \hat{v}^l)$ was chosen arbitrarily, it holds that $\rho_z(U(\mathcal{E} \setminus z))$ is nonblocking. Since renaming preserves nonblocking, it holds that $U(\mathcal{E} \setminus z)$ is nonblocking, which implies that $\mathcal{E} \setminus z$ is nonblocking.

Conversely assume $\mathcal{E} \setminus z$ is nonblocking. Then $U(\mathcal{E} \setminus z)$ is nonblocking, which implies $\rho_z(U(\mathcal{E} \setminus z))$ is nonblocking. It will be shown that $U(\mathcal{E})$ is nonblocking. Assume $(x_1^0, \ldots, x_n^0, \hat{v}^0) \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_m} (x_1^l, \ldots, x_n^l, \hat{v}^l)$ in $U(\mathcal{E})$. From Lemma 15, it follows that $(a^0, x_1^0, \ldots, x_n^0, \check{v}^0) \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_m} (a^l, x_1^l, \ldots, x_n^l, \check{v}^l)$ in $\rho_z(U(\mathcal{E} \setminus z))$, where $\hat{v}^i = \check{v}^i \oplus \{z \mapsto a^i\}$ for $0 \le i \le l$. Since $\rho_z(U(\mathcal{E} \setminus z))$ is nonblocking, there exists a path $(a^l, x_1^l, \ldots, x_n^l, \check{v}^l) \xrightarrow{\sigma_{l+1}} \cdots \xrightarrow{\sigma_m} (a^m, x_1^m, \ldots, x_n^m, \check{v}^m)$ in $\rho_z(U(\mathcal{E} \setminus z))$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. From Lemma 15 it follows that $(x_1^l, \ldots, x_n^l, \check{v}^l \oplus \{z \mapsto a^l\}) \xrightarrow{\sigma_{l+1}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \check{v}^m \oplus \{z \mapsto a^m\})$ in $U(\mathcal{E})$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times$

$\cdots \times Q_n^\omega$. Since $(x_1^l, \ldots, x_n^l, \hat{v}^l)$ was chosen arbitrarily, it follows that $U(\mathscr{E})$ is nonblocking, which implies that $\mathscr{E}$ is nonblocking. □

B.5 Event simplification

This section contains proofs of correctness of the event removal and merging operations in Propositions 9–12 in Section 5.5. The common approach to prove that abstractions such as these preserve the nonblocking property of an EFSM system, is to show that for each path in the EFSM system before abstraction there exists a corresponding path after abstraction, and vice versa.

First, to prove Prop. 9, which states that *false*-removal preserves the nonblocking property, it is shown in Lemma 16 that every path in any EFSM system resulting from restriction can be lifted to a path in the original system, and conversely it is shown in Lemma 17 that paths in an EFSM system also exist in a system resulting from *false*-removal.

Lemma 16 *Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system, let $\Omega \subseteq \Sigma_{\mathscr{E}}$, and let $\hat{u} \in dom(vars(\mathscr{E}) \setminus vars(\mathscr{E}_{|\Omega}))$. Then $(x_1, \ldots, x_n, \hat{v}) \stackrel{\sigma}{\to} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{E}_{|\Omega})$ implies $(x_1, \ldots, x_n, \hat{v} \oplus \hat{u}) \stackrel{\sigma}{\to} (y_1, \ldots, y_n, \hat{w} \oplus \hat{u})$ in $U(\mathscr{E})$.*

Proof Let $\mathscr{F} = \mathscr{E}_{|\Omega}$ and $V = \mathrm{vars}(\mathscr{E})$ and $W = \mathrm{vars}(\mathscr{E}_{|\Omega}) \subseteq \mathrm{vars}(\mathscr{E}) = V$.

Assume $(x_1, \ldots, x_n, \hat{v}) \stackrel{\sigma}{\to} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{F}) = U(\overset{\bullet}{\|}\mathscr{F})$. Then $\sigma \in \Omega$, and by Definition 10 it holds that $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\overset{\bullet}{\|}\mathscr{F}$ with $p \equiv \Delta_{\mathscr{F}}(\sigma)$ and $\Xi_W(p)(\hat{v}, \hat{w}) = \mathbf{T}$. By Definition 14, it holds that $x_i \xrightarrow{\sigma:p} y_i$ in $E_{i|\Omega}$ for each i such that $1 \leq i \leq n$ and $\sigma \in \Sigma_{E_i}$, with $p \equiv \Delta_{\mathscr{F}}(\sigma)$ and $\mathrm{vars}(p) \subseteq \mathrm{vars}(\mathscr{F}) = \mathrm{vars}(\mathscr{E}_{|\Omega}) = W$. As $\to_{|\Omega} \subseteq \to$, it follows that $x_i \xrightarrow{\sigma:p} y_i$ in E_i for each i such that $\sigma \in \Sigma_{E_i}$. This shows $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\overset{\bullet}{\|}\mathscr{E}$ by Definition 14. As furthermore $\Xi_W(p)(\hat{v}, \hat{w}) = \mathbf{T}$ and $\mathrm{vars}'(p) \subseteq \mathrm{vars}(p) \subseteq W$, it holds that $\Xi_V(p)(\hat{v} \oplus \hat{u}, \hat{w} \oplus \hat{u}) = \mathbf{T}$. Thus, $(x_1, \ldots, x_n, \hat{v} \oplus \hat{u}) \stackrel{\sigma}{\to} (y_1, \ldots, y_n, \hat{w} \oplus \hat{u})$ in $U(\overset{\bullet}{\|}\mathscr{E}) = U(\mathscr{E})$ by Definition 10. □

Lemma 17 *Let $\mathscr{E}$ be a normalised EFSM system, and let $\Lambda \subseteq \Sigma_{\mathscr{E}}$ be a set of events such that for all $\lambda \in \Lambda$ at least one of the following conditions holds:*

(i) *$\Delta_{\mathscr{E}}(\lambda) \equiv false$;*

(ii) *There exists $E \in \mathscr{E}$ such that $\lambda \in \Sigma_E$, but there does not exist any transition $x \xrightarrow{\lambda:p} y$ in E.*

Also let $W = vars(\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$. Then $(x_1, \ldots, x_n, \hat{v}) \stackrel{\sigma}{\to} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{E})$ implies $(x_1, \ldots, x_n, \hat{v}_{|W}) \stackrel{\sigma}{\to} (y_1, \ldots, y_n, \hat{w}_{|W})$ in $U(\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$.

Proof Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ and $\Omega = \Sigma_{\mathscr{E}} \setminus \Lambda$ and $\mathscr{F} = \mathscr{E}_{|\Omega}$ and $V = \mathrm{vars}(\mathscr{E})$. It is clear that $W = \mathrm{vars}(\mathscr{E}_{|\Omega}) \subseteq \mathrm{vars}(\mathscr{E}) = V$.

Assume $(x_1, \ldots, x_n, \hat{v}) \stackrel{\sigma}{\to} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{E}) = U(\overset{\bullet}{\|}\mathscr{E})$. By Definition 10, it follows that $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\overset{\bullet}{\|}\mathscr{E}$ with $p \equiv \Delta_{\mathscr{E}}(\sigma)$ and $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$. Note that $\sigma \in \Lambda$ cannot hold, because if $\sigma \in \Lambda$, then either (i) $p \equiv \Delta_{\mathscr{E}}(\sigma) \equiv false$ in contradiction to $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$, or (ii) there exists $E = E_k \in \mathscr{E}$ such that $\sigma \in \Sigma_E$ and $x_k \xrightarrow{\sigma:p} y_k$ in $E = E_k$ does not hold, in contradiction to $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\overset{\bullet}{\|}\mathscr{E}$ by Definition 9. Thus $\sigma \in \Omega$ and $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\overset{\bullet}{\|}\mathscr{E}_{|\Omega} = \overset{\bullet}{\|}\mathscr{F}$.

As also $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$ and $\mathrm{vars}'(p) \subseteq \mathrm{vars}(p) \subseteq \mathrm{vars}(\mathscr{E}_{|\Omega}) = W$, it follows that $\Xi_W(p)(\hat{v}_{|W}, \hat{w}_{|W}) = \mathbf{T}$. Thus, by Definition 10 it holds that $(x_1, \ldots, x_n, \hat{v}_{|W}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w}_{|W})$ in $U(\dot{\|}\mathscr{F}) = U(\mathscr{F}) = U(\mathscr{E}_{|\Omega})$. □

Proposition 9 (false-Removal) *Let $\mathscr{E}$ be a normalised EFSM system, and let $\Lambda \subseteq \Sigma_{\mathscr{E}}$ be a set of events such that for all $\lambda \in \Lambda$ at least one of the following conditions holds:*

(i) $\Delta_{\mathscr{E}}(\lambda) \equiv$ *false;*

(ii) *There exists $E \in \mathscr{E}$ such that $\lambda \in \Sigma_E$, but there does not exist any transition $x \xrightarrow{\lambda:p} y$ in E.*

Proof Note that $\mathscr{E}$ and thus $\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda}$ are normalised, so $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$ and $U(\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda}) = U(\dot{\|}\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$ by Proposition 1.

Assume $\mathscr{E}$ is nonblocking, which means that $U(\mathscr{E})$ is nonblocking. It will be shown that $U(\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$ is nonblocking. Let $(x\circ_1, \ldots, x_n^\circ, \hat{v}^\circ) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1, \hat{v}^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x_1^l, \ldots, x_n^l, \hat{v}^l)$ in $U(\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda}) = U(\dot{\|}\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$ where $(x_1^\circ, \ldots, x_n^\circ) \in Q_1^\circ \times \cdots \times Q_n^\circ$. By Lemma 16, it follows that $(x_1^0, \ldots, x_n^0, \hat{v} \circ \oplus \hat{u}^\circ) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1, \hat{v}^1 \oplus \hat{u}^\circ) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x_1^l, \ldots, x_n^l, \hat{v}^l \oplus \hat{u}^\circ)$ in $U(\dot{\|}\mathscr{E})$. Since $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$ is nonblocking, there exists a path $(x_1^l, \ldots, x_n^l, \hat{v}^l \oplus \hat{u}^\circ) \xrightarrow{\sigma_{l+1}} (x_1^{l+1}, \ldots, x_n^{l+1}, \hat{w}^{l+1}) \xrightarrow{\sigma_{l+2}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \hat{w}^m)$ in $U(\dot{\|}\mathscr{E})$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. From Lemma 17 and as $(\hat{v}^l \oplus \hat{u}\circ)_{|W} = \hat{v}^l$, it follows that $(x_1^l, \ldots, x_n^l, \hat{v}^l) \xrightarrow{\sigma_{l+1}} (x_1^{l+1}, \ldots, x_n^{l+1}, \hat{w}_{|W}^{l+1}) \xrightarrow{\sigma_{l+2}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \hat{v}_{|W}^m)$ in $U(\dot{\|}\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$ and $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. Since $(x_1^l, \ldots, x_n^l, \hat{v}^l)$ was chosen arbitrarily, it follows that $U(\dot{\|}\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda}) = U(\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$ is nonblocking, i.e., $\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda}$ is nonblocking.

Conversely assume $\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda}$ is nonblocking, which means that $U(\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$ is nonblocking. Let $(x_1^\circ, \ldots, x_n^\circ, \hat{v}^\circ) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1, \hat{v}^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x_1^l, \ldots, x_n^l, \hat{v}^l)$ in $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$ where $(x_1^\circ, \ldots, x_n^\circ) \in Q_1^\circ \times \cdots \times Q_n^\circ$. By Lemma 17, it holds that $(x_1^\circ, \ldots, x_n^\circ, \hat{v}_{|W}^\circ) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1, \hat{v}_{|W}^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x_1^l, \ldots, x_n^l, \hat{v}_{|W}^l)$ in $U(\dot{\|}\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$. As $U(\dot{\|}\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda}) = U(\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$ is nonblocking, there exists a path $(x_1^l, \ldots, x_n^l, \hat{v}_{|W}^l) \xrightarrow{\sigma_{l+1}} (x_1^{l+1}, \ldots, x_n^{l+1}, \hat{w}^{l+1}) \xrightarrow{\sigma_{l+2}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \hat{w}^m)$ in $U(\dot{\|}\mathscr{E}_{|\Sigma_{\mathscr{E}} \setminus \Lambda})$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. By Lemma 16, it follows that $(x_1^l, \ldots, x_n^l, \hat{v}^l) = (x_1^l, \ldots, x_n^l, \hat{v}_{|W}^l \oplus \hat{v}_{|V \setminus W}^l) \xrightarrow{\sigma_{l+1}} (x_1^{l+1}, \ldots, x_n^{l+1}, \hat{w}^{l+1} \oplus \hat{v}_{|V \setminus W}^l) \xrightarrow{\sigma_{l+2}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \hat{w}^m \oplus \hat{v}_{|V \setminus W}^l)$ in $U(\dot{\|}\mathscr{E})$ and $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. As $(x_1^l, \ldots, x_n^l, \hat{v}^l)$ was chosen arbitrarily, it follows that $U(\dot{\|}\mathscr{E}) = U(\mathscr{E})$ is nonblocking, i.e., $\mathscr{E}$ is nonblocking. □

As selfloop removal is also defined using restriction, the proof of Proposition 10 again uses Lemma 16 to lift paths from an abstracted system to the original system. For the converse, the following Lemma 18 shows that a path in the original system also exists in the abstracted system after selfloop removal, except possibly for the deletion of some selfloops.

Lemma 18 *Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system with event alphabet $\Sigma_{\mathscr{E}} = \Omega \dot{\cup} \Lambda$, which is selfloop-only for Λ. Then $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma}$*

$(y_1, \ldots, y_n, \hat{w})$ in $U(\mathcal{E})$ implies $(x_1, \ldots, x_n, \hat{v}_{|W}) \xrightarrow{P_\Omega(\sigma)} (y_1, \ldots, y_n, \hat{w}_{|W})$ in $U(\mathcal{E}_{|\Omega})$ where $W = vars(\mathcal{E}_{|\Omega})$.

Proof Let $V = \mathrm{vars}(\mathcal{E})$. Clearly $W = \mathrm{vars}(\mathcal{E}_{|\Omega}) \subseteq \mathrm{vars}(\mathcal{E}) = V$.

Assume that $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathcal{E}) = U(\dot{\|}\mathcal{E})$. Then by Definition 10 it holds that $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\dot{\|}\mathcal{E}$ with $p \equiv \Delta_{\mathcal{E}}(\sigma)$ and $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$. By Definition 14 it holds that $x_i \xrightarrow{\sigma:p} y_i$ for each i such that $1 \le i \le n$ and $\sigma \in \Sigma_{E_i}$, and $x_i = y_i$ for each i such that $1 \le i \le n$ and $\sigma \notin \Sigma_{E_i}$. Consider two cases for σ: either $\sigma \in \Lambda$ or $\sigma \notin \Lambda$.

- If $\sigma \in \Lambda$ then $P_\Omega(\sigma) = \varepsilon$, and since each E_i is selfloop-only for $\sigma \in \Lambda$, it follows from $x_i \xrightarrow{\sigma:p} y_i$ in E_i that $x_i = y_i$ and $\mathrm{vars}'(p) = \emptyset$. From $\mathrm{vars}'(p) = \emptyset$ and $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$ it follows that $\hat{v} = \hat{w}$, which implies $\hat{v}_{|W} = \hat{w}_{|W}$. Given $P_\Omega(\sigma) = \varepsilon$, it follows that $(x_1, \ldots, x_n, \hat{v}_{|W}) \xrightarrow{P_\Omega(\sigma)} (x_1, \ldots, x_n, \hat{v}_{|W}) = (y_1, \ldots, y_n, \hat{w}_{|W})$ in $U(\dot{\|}\mathcal{E}_{|\Omega}) = U(\mathcal{E}_{|\Omega})$.
- If $\sigma \notin \Lambda$ then $P_\Omega(\sigma) = \sigma$. In this case, it follows from $x_i \xrightarrow{\sigma:p} y_i$ in E_i that $x_i \xrightarrow{\sigma:p} y_i$ in $E_{i\,|\Omega}$, and thus $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\dot{\|}\mathcal{E}_{|\Omega}$. As this transition is in $\dot{\|}\mathcal{E}_{|\Omega}$, it holds that $\mathrm{vars}'(p) \subseteq \mathrm{vars}(p) \subseteq \mathrm{vars}(\dot{\|}\mathcal{E}_{|\Omega}) = \mathrm{vars}(\mathcal{E}_{|\Omega}) = W$, so it follows from $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$ that $\Xi_W(p)(\hat{v}_{|W}, \hat{w}_{|W}) = \mathbf{T}$. By Definition 10, it follows that $(x_1, \ldots, x_n, \hat{v}_{|W}) \xrightarrow{P_\Omega(\sigma)} (y_1, \ldots, y_n, \hat{w}_{|W})$ in $U(\dot{\|}\mathcal{E}_{|\Omega}) = U(\mathcal{E}_{|\Omega})$.

□

Proposition 10 (Selfloop Removal) *Let $\mathcal{E}$ be a normalised EFSM system that is selfloop-only for $\Lambda \subseteq \Sigma_{\mathcal{E}}$. Then $\mathcal{E}$ is nonblocking if and only if $\mathcal{E}_{|\Sigma_{\mathcal{E}} \setminus \Lambda}$ is nonblocking.*

Proof Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ and $\Omega = \Sigma_{\mathcal{E}} \setminus \Lambda$ and $V = \mathrm{vars}(\mathcal{E})$ and $W = \mathrm{vars}(\mathcal{E}_{|\Omega})$.

Assume $\mathcal{E}$ is nonblocking, which means that $U(\mathcal{E})$ is nonblocking. It will be shown that $U(\mathcal{E}_{|\Omega})$ is nonblocking. Let $(x_1^\circ, \ldots, x_n^\circ, \hat{v}^\circ) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1, \hat{v}^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x_1^l, \ldots, x_n^l, \hat{v}^l)$ in $U(\mathcal{E}_{|\Omega}) = U(\dot{\|}\mathcal{E}_{|\Omega})$ where $(x_1^\circ, \ldots, x_n^\circ) \in Q_1^\circ \times \cdots \times Q_n^\circ$. By Lemma 16, it follows that $(x_1^0, \ldots, x_n^0, \hat{v} \;\circ\; \oplus \hat{u}^\circ) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1, \hat{v}^1 \oplus \hat{u}^\circ) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x_1^l, \ldots, x_n^l, \hat{v}^l \oplus \hat{u}^\circ)$ in $U(\dot{\|}\mathcal{E})$. Since $U(\mathcal{E}) = U(\dot{\|}\mathcal{E})$ is nonblocking, there exists a path $(x_1^l, \ldots, x_n^l, \hat{v}^l \oplus \hat{u}^\circ) \xrightarrow{\sigma_{l+1}} (x_1^{l+1}, \ldots, x_n^{l+1}, \hat{w}^{l+1}) \xrightarrow{\sigma_{l+2}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \hat{w}^m)$ in $U(\dot{\|}\mathcal{E})$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. From Lemma 18 and since $(\hat{v}^l \oplus \hat{u}^\circ)_{|W} = \hat{v}^l$, it follows that $(x_1^l, \ldots, x_n^l, \hat{v}^l) \xrightarrow{P_\Omega(\sigma_{l+1})} (x_1^{l+1}, \ldots, x_n^{l+1}, \hat{w}_{|W}^{l+1}) \xrightarrow{P_\Omega(\sigma_{l+2})} \cdots \xrightarrow{P_\Omega(\sigma_m)} (x_1^m, \ldots, x_n^m, \hat{v}_{|W}^m)$ in $U(\dot{\|}\mathcal{E}_{|\Omega})$ and $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. Since $(x_1^l, \ldots, x_n^l, \hat{v}^l)$ was chosen arbitrarily, it follows that $U(\dot{\|}\mathcal{E}_{|\Omega}) = U(\mathcal{E}_{|\Omega})$ is nonblocking, i.e., $\mathcal{E}_{|\Omega}$ is nonblocking.

Conversely assume $\mathcal{E}_{|\Omega}$ is nonblocking, which means that $U(\mathcal{E}_{|\Omega})$ is nonblocking. Let $(x_1^\circ, \ldots, x_n^\circ, \hat{v}^\circ) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1, \hat{v}^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x_1^l, \ldots, x_n^l, \hat{v}^l)$ in $U(\mathcal{E}) = U(\dot{\|}\mathcal{E})$ where $(x{\circ}_1, \ldots, x{\circ}_n) \in Q_1^\circ \times \cdots \times Q_n^\circ$. By Lemma 18, it holds that $(x_1^\circ, \ldots, x_n^\circ, \hat{v}_{|W}^\circ) \xrightarrow{P_\Omega(\sigma_1)} (x_1^1, \ldots, x_n^1, \hat{v}_{|W}^1) \xrightarrow{P_\Omega(\sigma_2)} \cdots \xrightarrow{P_\Omega(\sigma_l)} (x_1^l, \ldots, x_n^l, \hat{v}_{|W}^l)$ in $U(\dot{\|}\mathcal{E}_{|\Omega})$. Since $U(\dot{\|}\mathcal{E}_{|\Omega}) = U(\mathcal{E}_{|\Omega})$ is nonblocking, there exists a path $(x_1^l, \ldots, x_n^l, \hat{v}_{|W}^l) \xrightarrow{\sigma_{l+1}} (x_1^{l+1}, \ldots, x_n^{l+1}, \hat{w}^{l+1}) \xrightarrow{\sigma_{l+2}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \hat{w}^m)$ in $U(\dot{\|}\mathcal{E}_{|\Omega})$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. By Lemma 16, it follows that $(x_1^l, \dots, x_n^l, \hat{v}^l) = (x_1^l, \dots, x_n^l, \hat{v}^l_{|W} \oplus \hat{v}^l_{|V\setminus W}) \xrightarrow{\sigma_{l+1}} (x_1^{l+1}, \dots, x_n^{l+1}, \hat{w}^{l+1} \oplus \hat{v}^l_{|V\setminus W}) \xrightarrow{\sigma_{l+2}} \cdots \xrightarrow{\sigma_m} (x_1^m, \dots, x_n^m, \hat{w}^m \oplus \hat{v}^l_{|V\setminus W})$ in $U(\dot{\|}\mathscr{E})$ and $(x_1^m, \dots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. As $(x_1^l, \dots, x_n^l, \hat{v}^l)$ was chosen arbitrarily, it follows that $U(\dot{\|}\mathscr{E}) = U(\mathscr{E})$ is nonblocking, i.e., $\mathscr{E}$ is nonblocking. □

The next result to prove is Proposition 11, which states that the nonblocking property of an EFSM system is preserved by event merging. Again, it needs to be established that for each path in the EFSM system before abstraction there exists a corresponding path after abstraction, and vice versa. First, Lemma 19 shows that every path in an EFSM system can be found again after renaming, and afterwards Lemma 20 shows how to lift a path from the abstracted system after event merging back to the original system.

Lemma 19 *Let $\mathscr{E} = \{E_1, \dots, E_n\}$ be an EFSM system, and let $\rho\colon \Sigma_\mathscr{E} \to \Sigma'$ be an arbitrary renaming. Then $(x_1, \dots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \dots, y_n, \hat{w})$ in $U(\mathscr{E})$ implies $(x_1, \dots, x_n, \hat{v}) \xrightarrow{\rho(\sigma)} (y_1, \dots, y_n, \hat{w})$ in $U(\rho(\mathscr{E}))$.*

Proof Write $V = \text{vars}(E)$ and $\Sigma_i = \Sigma_{E_i}$ for $1 \le i \le n$. Assume $(x_1, \dots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \dots, y_n, \hat{w})$ in $U(\mathscr{E})$. By Definition 10, this means $(x_1, \dots, x_n) \xrightarrow{\sigma:p} (y_1, \dots, y_n)$ in $\|\mathscr{E}$ where $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$. By Definition 9, it holds that $x_i \xrightarrow{\sigma:p_i} y_i$ for each E_i such that $\sigma \in \Sigma_i$, and $x_i = y_i$ for each E_i such that $\sigma \notin \Sigma_i$, and $p \equiv \bigwedge_{\sigma\in\Sigma_i} p_i$. For $\sigma \in \Sigma_i$ it follows that $x_i \xrightarrow{\rho(\sigma):p_i} y_i$ in $\rho(E_i)$, and therefore $(x_1, \dots, x_n) \xrightarrow{\rho(\sigma):p} (y_1, \dots, y_n)$ in $\|\rho(\mathscr{E})$. As $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$ and $\text{vars}(\rho(\mathscr{E})) = \text{vars}(\mathscr{E}) = V$, it follows by Definition 10 that $(x_1, \dots, x_n, \hat{v}) \xrightarrow{\rho(\sigma)} (y_1, \dots, y_n, \hat{w})$ in $U(\rho(\mathscr{E}))$. □

Lemma 20 *Let $\mathscr{E} = \{E_1, \dots, E_n\}$ be a normalised EFSM system with $E_i = \langle \Sigma_i, Q_i, \to_i, Q^\circ_\omega, Q_i^\omega \rangle$, let $E_k \in \mathscr{E}$, and let $\rho\colon \Sigma_\mathscr{E} \to \Sigma'$ be a renaming such that the following conditions hold for all $\sigma_1, \sigma_2 \in \Sigma_\mathscr{E}$ with $\rho(\sigma_1) = \rho(\sigma_2)$:*

(i) *$\Delta_\mathscr{E}(\sigma_1) = \Delta_\mathscr{E}(\sigma_2)$;*
(ii) *for all $i \neq k$, it holds that $\sigma_1 \in \Sigma_i$ if and only if $\sigma_2 \in \Sigma_i$, and for all $x, y \in Q_i$ it holds that $x \xrightarrow{\sigma_1:\Delta_\mathscr{E}(\sigma_1)}_i y$ if and only if $x \xrightarrow{\sigma_2:\Delta_\mathscr{E}(\sigma_2)}_i y$.*

Then $(x_1, \dots, x_n, \hat{v}) \xrightarrow{\mu} (y_1, \dots, y_n, \hat{w})$ in $U(\rho(\mathscr{E}))$ implies $(x_1, \dots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \dots, y_n, \hat{w})$ in $U(\mathscr{E})$ for some $\sigma \in \Sigma_\mathscr{E}$ such that $\rho(\sigma) = \mu$.

Proof First note that $\mathscr{E}$ is normalised, which implies by assumption (i) that $\rho(\mathscr{E})$ is normalised. Therefore, it holds by Proposition 1 that $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$ and $U(\rho(\mathscr{E})) = U(\dot{\|}\rho(\mathscr{E}))$.

Assume $(x_1, \dots, x_n, \hat{v}) \xrightarrow{\mu} (y_1, \dots, y_n, \hat{w})$ in $U(\rho(\mathscr{E})) = U(\dot{\|}\rho(\mathscr{E}))$. Then it holds by Definition 10 that $(x_1, \dots, x_n) \xrightarrow{\mu:p} (y_1, \dots, y_n)$ in $\dot{\|}\rho(\mathscr{E})$ where $p \equiv \Delta_{\rho(\mathscr{E})}(\mu)$ and $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$ where $V = \text{vars}(\rho(\mathscr{E})) = \text{vars}(\mathscr{E})$. Consider two cases: either $\mu \in \rho(\Sigma_k)$ or $\mu \notin \rho(\Sigma_k)$.

- If $\mu \in \rho(\Sigma_k)$, then $x_k \xrightarrow{\mu:p} y_k$ in $\rho(E_k)$ by Definition 14. Then there exists $\sigma \in \Sigma_k$ such that $\rho(\sigma) = \mu$ and $x_k \xrightarrow{\sigma:p} y_k$ in E_k.
- If $\mu \notin \rho(\Sigma_k)$, then $x_k = y_k$ by Definition 14. As ρ is surjective by Definition 15, there exists $\sigma \in \Sigma_\mathscr{E}$ such that $\rho(\sigma) = \mu$. Note that $\sigma \notin \Sigma_k$ as otherwise $\mu = \rho(\sigma) \in \rho(\Sigma_k)$.

In both cases there exists $\sigma \in \Sigma_{\mathcal{E}}$ with $\rho(\sigma) = \mu$, with other properties mentioned in each case. Now consider two cases for each $i \neq k$: either $\sigma \in \Sigma_i$ or $\sigma \notin \Sigma_i$.

- If $\sigma \in \Sigma_i$, then $\mu = \rho(\sigma) \in \rho(\Sigma_i)$ and thus $x_i \xrightarrow{\mu:p} y_i$ in $\rho(E_i)$ by Definition 14. Then there exists $\sigma_i \in \Sigma_i$ such that $\rho(\sigma_i) = \mu$ and $x_i \xrightarrow{\sigma_i:p} y_i$ in E_i. As $i \neq k$ and $\rho(\sigma_i) = \mu = \rho(\sigma)$, it follows by assumption (ii) that $\sigma \in \Sigma_i$ and $x_i \xrightarrow{\sigma:p} y_i$ in E_i.
- If $\sigma \notin \Sigma_i$, then $\mu = \rho(\sigma) \notin \rho(\Sigma_i)$ and thus $x_i = y_i$ by Definition 14.

Combining the above observations for k and all $i \neq k$, it follows by Definition 14 that $(x_1, \ldots, x_n) \xrightarrow{\sigma:p} (y_1, \ldots, y_n)$ in $\|\mathcal{E}$. As furthermore $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$, it follows by Definition 10 that $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\|\mathcal{E}) = U(\mathcal{E})$. □

Proposition 11 (Event Merging) *Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system with $E_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q^{\circ}_{\omega}, Q^{\omega}_i \rangle$, let $E_k \in \mathcal{E}$, and let $\rho\colon \Sigma_{\mathcal{E}} \rightarrow \Sigma'$ be a renaming such that the following conditions hold for all $\sigma_1, \sigma_2 \in \Sigma_{\mathcal{E}}$ with $\rho(\sigma_1) = \rho(\sigma_2)$:*

(i) *$\Delta_{\mathcal{E}}(\sigma_1) = \Delta_{\mathcal{E}}(\sigma_2)$;*
(ii) *for all $i \neq k$, it holds that $\sigma_1 \in \Sigma_i$ if and only if $\sigma_2 \in \Sigma_i$, and for all $x, y \in Q_i$ it holds that $x \xrightarrow{\sigma_1:\Delta_{\mathcal{E}}(\sigma_1)}_i y$ if and only if $x \xrightarrow{\sigma_2:\Delta_{\mathcal{E}}(\sigma_2)}_i y$.*

Then $\mathcal{E}$ is nonblocking if and only if $\rho(\mathcal{E})$ is nonblocking.

Proof First assume $\mathcal{E}$ is nonblocking, which means that $U(\mathcal{E})$ is nonblocking. It will be shown that $\mathcal{E}$ is nonblocking. Let $U(\rho(\mathcal{E})) \xrightarrow{\mu_1} (x^1_1, \ldots, x^1_n, \hat{v}^1) \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_l} (x^l_1, \ldots, x^l_n, \hat{v}^l)$. By Lemma 20, there exist events $\sigma_1, \ldots, \sigma_l$ such that $U(\mathcal{E}) \xrightarrow{\sigma_1} (x^1_1, \ldots, x^1_n, \hat{v}^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x^l_1, \ldots, x^l_n, \hat{v}^l)$. Since $U(\mathcal{E})$ is nonblocking, there exists a path $(x^l_1, \ldots, x^l_n, \hat{v}^l) \xrightarrow{\sigma_{l+1}} \cdots \xrightarrow{\sigma_m} (x^m_1, \ldots, x^m_n, \hat{v}^m)$ in $U(\mathcal{E})$ such that $(x^m_1, \ldots, x^m_n) \in Q^{\omega}[1] \times \cdots \times Q^{\omega}[n]$. From Lemma 19, it follows that $(x^l_1, \ldots, x^l_n, \hat{v}^l) \xrightarrow{\rho(\sigma_{l+1})} \cdots \xrightarrow{\rho(\sigma_m)} (x^m_1, \ldots, x^m_n, \hat{v}^m)$ in $U(\rho(\mathcal{E}))$ and $(x^m_1, \ldots, x^m_n) \in Q^{\omega}_1 \times \cdots \times Q^{\omega}_n$. Since $(x^l_1, \ldots, x^l_n, \hat{v}^l)$ was chosen arbitrarily, it follows that $U(\rho(\mathcal{E}))$ is nonblocking, i.e., $\mathcal{E}$ is nonblocking.

Conversely assume $\mathcal{F}$ is nonblocking, which means that $U(\mathcal{F})$ is nonblocking. Let $U(\mathcal{E}) \xrightarrow{\sigma_1} (x^1_1, \ldots, x^1_n, \hat{v}^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x^l_1, \ldots, x^l_n, \hat{v}^l)$. By Lemma 19, it holds that $U(\mathcal{F}) \xrightarrow{\rho(\sigma_1)} (x^1_1, \ldots, x^1_n, \hat{v}^1) \xrightarrow{\rho(\sigma_2)} \cdots \xrightarrow{\rho(\sigma_l)} (x^l_1, \ldots, x^l_n, \hat{v}^l)$. As $U(\mathcal{F})$ is nonblocking, there exists a path $(x^l_1, \ldots, x^l_n, \hat{v}^l) \xrightarrow{\mu_{l+1}} \cdots \xrightarrow{\mu_m} (x^m_1, \ldots, x^m_n, \hat{v}^m)$ in $U(\mathcal{F})$ such that $(x^m_1, \ldots, x^m_n) \in Q^{\omega}_1 \times \cdots \times Q^{\omega}_n$. By Lemma 20, there exist events $\sigma_{1+1}, \ldots, \sigma_m$ such that $(x^l_1, \ldots, x^l_n, \hat{v}^l) \xrightarrow{\sigma_{l+1}} \cdots \xrightarrow{\sigma_m} (x^m_1, \ldots, x^m_n, \hat{v}^m)$ in $U(\mathcal{E})$ and $(x^m_1, \ldots, x^m_n) \in Q^{\omega}_1 \times \cdots \times Q^{\omega}_n$. As $(x^l_1, \ldots, x^l_n, \hat{v}^l)$ was chosen arbitrarily, it follows that $U(\mathcal{E})$ is nonblocking, i.e., $\mathcal{E}$ is nonblocking. □

Similar to event merging, to prove that update merging preserves the nonblocking property of an EFSM system as stated in Proposition 12, the relationship between the paths in the system before and after abstraction is first established. Lemma 21 shows how to construct a path in the abstracted system after update merging from a path in the original system, and Lemma 22 shows how to do the converse.

Lemma 21 *Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system with*

$E_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q^\circ_\omega, Q^\omega_i \rangle$. Let ρ be a renaming such that the following conditions hold for all $\sigma_1, \sigma_2 \in \Sigma_{\mathscr{E}}$ with $\rho(\sigma_1) = \rho(\sigma_2)$:

(i) *$\mathit{vars}'(\Delta_{\mathscr{E}}(\sigma_1)) = \mathit{vars}'(\Delta_{\mathscr{E}}(\sigma_2))$,*
(ii) *for all $i = 1, \ldots, n$ it holds that $\sigma_1 \in \Sigma_i$ if and only if $\sigma_2 \in \Sigma_i$, and for all $x, y \in Q_i$ it holds that $x \xrightarrow{\sigma_1 : \Delta_{\mathscr{E}}(\sigma_1)}_i y$ if and only if $x \xrightarrow{\sigma_2 : \Delta_{\mathscr{E}}(\sigma_2)}_i y$*

Further let $\mathscr{F} = \{F_1, \ldots, F_n\}$ such that $F_i = \langle \rho(\Sigma_i), Q_i, \rightarrow^F_i, Q^\circ_i, Q^\omega_i \rangle$ where $\rightarrow^F_i = \{(x, \rho(\sigma), \Delta_{\mathscr{F}}(\rho(\sigma)), y) \mid x \xrightarrow{\sigma : \Delta_{\mathscr{E}}(\sigma)} y\}$ and $\Delta_{\mathscr{F}}(\mu) \equiv \bigvee_{\sigma \in \rho^{-1}(\mu)} \Delta_{\mathscr{E}}(\sigma)$ for all $\mu \in \Sigma_{\mathscr{F}}$. Then $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{E})$ implies $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\rho(\sigma)} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{F})$.

Proof Note that $\mathscr{E}$ and $\mathscr{F}$ are normalised, so $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$ and $U(\mathscr{F}) = U(\dot{\|}\mathscr{F})$ by Proposition 1.

Let $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$. This means by Definition 10 that $(x_1, \ldots, x_n) \xrightarrow{\sigma : \Delta_{\mathscr{E}}(\sigma)} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{E}$ and $\Xi_V(\Delta_{\mathscr{E}}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$ where $V = \mathrm{vars}(\mathscr{E}) = \mathrm{vars}(\mathscr{F})$. Consider two cases for each E_i: either $\sigma \in \Sigma_i$ or $\sigma \notin \Sigma_i$.

- If $\sigma \in \Sigma_i$, then it follows by Definition 14 that $x_i \xrightarrow{\sigma : \Delta_{\mathscr{E}}(\sigma)} y_i$ in E_i. In this case, $\rho(\sigma) \in \rho(\Sigma_i)$, and it follows by construction of $\rightarrow^F_i$ that $x_i \xrightarrow{\rho(\sigma) : \Delta_{\mathscr{F}}(\sigma)} y_i$ in F_i.
- If $\sigma \notin \Sigma_i$ then $x_i = y_i$ by Definition 14, and $\rho(\sigma) \notin \rho(\Sigma_i)$.

Combining these observations for all i, it follows by Definition 14 that $(x_1, \ldots, x_n) \xrightarrow{\rho(\sigma) : \Delta_{\mathscr{F}}(\rho(\sigma))} (y_1, \ldots, y_n)$ in $F_1 \dot{\|} \cdots \dot{\|} F_n = \dot{\|}\mathscr{F}$. Furthermore, note that by construction $\Delta_{\mathscr{F}}(\rho(\sigma)) = \bigvee_{\sigma' \in \rho^{-1}(\rho(\sigma))} \Delta_{\mathscr{E}}(\sigma')$, which implies $\Xi_V(\Delta_{\mathscr{F}}(\rho(\sigma))) \Leftrightarrow \bigvee_{\sigma' \in \rho^{-1}(\rho(\sigma))} \Xi_V(\Delta_{\mathscr{E}}(\sigma'))$ by assumption (i). Then it follows from $\sigma \in \rho^{-1}(\rho(\sigma))$ and $\Xi_V(\Delta_{\mathscr{E}}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$ that $\Xi_V(\Delta_{\mathscr{F}}(\rho(\sigma)))(\hat{v}, \hat{w}) = \mathbf{T}$. Then it follows from Definition 10 that $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\rho(\sigma)} (y_1, \ldots, y_n, \hat{w})$ in $U(\dot{\|}\mathscr{F}) = U(\mathscr{F})$. □

Lemma 22 *Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system with $E_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q^\circ_\omega, Q^\omega_i \rangle$. Let ρ be a renaming such that the following conditions hold for all $\sigma_1, \sigma_2 \in \Sigma_{\mathscr{E}}$ with $\rho(\sigma_1) = \rho(\sigma_2)$:*

(i) *$\mathit{vars}'(\Delta_{\mathscr{E}}(\sigma_1)) = \mathit{vars}'(\Delta_{\mathscr{E}}(\sigma_2))$,*
(ii) *for all $i = 1, \ldots, n$ it holds that $\sigma_1 \in \Sigma_i$ if and only if $\sigma_2 \in \Sigma_i$, and for all $x, y \in Q_i$ it holds that $x \xrightarrow{\sigma_1 : \Delta_{\mathscr{E}}(\sigma_1)}_i y$ if and only if $x \xrightarrow{\sigma_2 : \Delta_{\mathscr{E}}(\sigma_2)}_i y$*

Further let $\mathscr{F} = \{F_1, \ldots, F_n\}$ such that $F_i = \langle \rho(\Sigma_i), Q_i, \rightarrow^F_i, Q^\circ_i, Q^\omega_i \rangle$ where $\rightarrow^F_i = \{(x, \rho(\sigma), \Delta_{\mathscr{F}}(\rho(\sigma)), y) \mid x \xrightarrow{\sigma : \Delta_{\mathscr{E}}(\sigma)} y\}$ and $\Delta_{\mathscr{F}}(\mu) \equiv \bigvee_{\sigma \in \rho^{-1}(\mu)} \Delta_{\mathscr{E}}(\sigma)$ for all $\mu \in \Sigma_{\mathscr{F}}$. Then $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\mu} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{F})$ implies $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{E})$ for some $\sigma \in \Sigma_{\mathscr{E}}$ such that $\rho(\sigma) = \mu$.

Proof Note that $\mathscr{E}$ and $\mathscr{F}$ are normalised, so $U(\mathscr{E}) = U(\dot{\|}\mathscr{E})$ and $U(\mathscr{F}) = U(\dot{\|}\mathscr{F})$ by Proposition 1.

Assume $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\mu} (y_1, \ldots, y_n, \hat{w})$ in $U(\mathscr{F}) = U(\dot{\|}\mathscr{F})$. Then it holds that by Definition 10 that $(x_1, \ldots, x_n) \xrightarrow{\mu:p} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{F}$ where $p \equiv \Delta_{\mathscr{F}}(\mu) \equiv \bigvee_{\sigma \in \rho^{-1}(\mu)} \Delta_{\mathscr{E}}(\sigma)$, and $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$ where $V = \mathrm{vars}(\mathscr{F})$. As $p \equiv \bigvee_{\sigma \in \rho^{-1}(\mu)} \Delta_{\mathscr{E}}(\sigma)$ and $\Xi_V(p)(\hat{v}, \hat{w}) = \mathbf{T}$, there exists $\sigma \in \rho^{-1}(\mu)$ such that $\Xi_V(\Delta_{\mathscr{E}}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$. Note, as $\sigma \in \rho^{-1}(\mu)$ it holds that $\rho(\sigma) = \mu$. Consider two cases for each E_i: either $\sigma \in \Sigma_i$ or $\sigma \notin \Sigma_i$.

- If $\sigma \in \Sigma_i$ then $\mu = \rho(\sigma) \in \rho(\Sigma_i)$, which by Definition 14 implies $x_i \xrightarrow{\mu:p} y_i$ in F_i. By construction of $\rightarrow_i^F$, this means $x_i \xrightarrow{\sigma:\Delta_{\mathscr{E}}(\sigma)} y_i$ in E_i.
- If $\sigma \notin \Sigma_i$, then $\mu \notin \rho(\Sigma_i)$ and $x_i = y_i$ by Definition 14.

Combining the above observations for all i, it follows by Definition 14 that $(x_1, \ldots, x_n) \xrightarrow{\sigma:\Delta_{\mathscr{E}}(\sigma)} (y_1, \ldots, y_n)$ in $\dot{\|}\mathscr{E}$. As $\Xi_V(\Delta_{\mathscr{E}}(\sigma))(\hat{v}, \hat{w}) = \mathbf{T}$, it follows that $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w})$ in $U(\dot{\|}\mathscr{E}) = U(\mathscr{E})$. □

Proposition 12 (Update Merging) *Let $\mathscr{E} = \{E_1, \ldots, E_n\}$ be a normalised EFSM system with $E_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_\omega^\circ, Q_i^\omega \rangle$. Let ρ be a renaming such that the following conditions hold for all $\sigma_1, \sigma_2 \in \Sigma_{\mathscr{E}}$ with $\rho(\sigma_1) = \rho(\sigma_2)$:*

(i) *$\mathit{vars}'(\Delta_{\mathscr{E}}(\sigma_1)) = \mathit{vars}'(\Delta_{\mathscr{E}}(\sigma_2))$,*
(ii) *for all $i = 1, \ldots, n$ it holds that $\sigma_1 \in \Sigma_i$ if and only if $\sigma_2 \in \Sigma_i$, and for all $x, y \in Q_i$ it holds that $x \xrightarrow{\sigma_1:\Delta_{\mathscr{E}}(\sigma_1)}_i y$ if and only if $x \xrightarrow{\sigma_2:\Delta_{\mathscr{E}}(\sigma_2)}_i y$*

Further let $\mathscr{F} = \{F_1, \ldots, F_n\}$ such that $F_i = \langle \rho(\Sigma_i), Q_i, \rightarrow_i^F, Q_i^\circ, Q_i^\omega \rangle$ where $\rightarrow_i^F = \{(x, \rho(\sigma), \Delta_{\mathscr{F}}(\rho(\sigma)), y) \mid x \xrightarrow{\sigma:\Delta_{\mathscr{E}}(\sigma)} y\}$ and $\Delta_{\mathscr{F}}(\mu) \equiv \bigvee_{\sigma \in \rho^{-1}(\mu)} \Delta_{\mathscr{E}}(\sigma)$ for all $\mu \in \Sigma_{\mathscr{F}}$. Then $\mathscr{E}$ is nonblocking if and only if $\mathscr{F}$ is nonblocking.

Proof First assume $\mathscr{E}$ is nonblocking, which means that $U(\mathscr{E})$ is nonblocking. It will be shown that $\mathscr{F}$ is nonblocking. Let $U(\mathscr{F}) \xrightarrow{\mu_1} (x_1^1, \ldots, x_n^1, \hat{v}^1) \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_l} (x_1^l, \ldots, x_n^l, \hat{v}^l)$. By Lemma 22, there exist events $\sigma_1, \ldots, \sigma_l$ such that $U(\mathscr{E}) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1, \hat{v}^1) \xrightarrow{\sigma_2} \cdots\ \mathit{underrightarrow}_{\sigma_l} (x_1^l, \ldots, x_n^l, \hat{v}^l)$. Since $U(\mathscr{E})$ is nonblocking, there exists a path $(x_1^l, \ldots, x_n^l, \hat{v}^l) \xrightarrow{\sigma_{l+1}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \hat{v}^m)$ in $U(\mathscr{E})$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. From Lemma 21, it follows that $(x_1^l, \ldots, x_n^l, \hat{v}^l) \xrightarrow{\rho(\sigma_{l+1})} \cdots \xrightarrow{\rho(\sigma_m)} (x_1^m, \ldots, x_n^m, \hat{v}^m)$ in $U(\mathscr{F})$ and $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. Since $(x_1^l, \ldots, x_n^l, \hat{v}^l)$ was chosen arbitrarily, it follows that $U(\mathscr{F})$ is nonblocking, i.e., $\mathscr{F}$ is nonblocking.

Conversely assume $\mathscr{F}$ is nonblocking, which means that $U(\mathscr{F})$ is nonblocking. Let $U(\mathscr{E}) \xrightarrow{\sigma_1} (x_1^1, \ldots, x_n^1, \hat{v}^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_l} (x_1^l, \ldots, x_n^l, \hat{v}^l)$. By Lemma 21, it holds that $U(\mathscr{F}) \xrightarrow{\rho(\sigma_1)} (x_1^1, \ldots, x_n^1, \hat{v}^1) \xrightarrow{\rho(\sigma_2)} \cdots \xrightarrow{\rho(\sigma_l)} (x_1^l, \ldots, x_n^l, \hat{v}^l)$. As $U(\mathscr{F})$ is nonblocking, there exists a path $(x_1^l, \ldots, x_n^l, \hat{v}^l) \xrightarrow{\mu_{l+1}} \cdots \xrightarrow{\mu_m} (x_1^m, \ldots, x_n^m, \hat{v}^m)$ in $U(\mathscr{F})$ such that $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. By Lemma 22, there exist events $\sigma_{l+1}, \ldots, \sigma_m$ such that $(x_1^l, \ldots, x_n^l, \hat{v}^l) \xrightarrow{\sigma_{l+1}} \cdots \xrightarrow{\sigma_m} (x_1^m, \ldots, x_n^m, \hat{v}^m)$ in $U(\mathscr{E})$ and $(x_1^m, \ldots, x_n^m) \in Q_1^\omega \times \cdots \times Q_n^\omega$. As $(x_1^l, \ldots, x_n^l, \hat{v}^l)$ was chosen arbitrarily, it follows that $U(\mathscr{E})$ is nonblocking, i.e., $\mathscr{E}$ is nonblocking. □

References

Åkesson K, Fabian M, Flordal H, Malik R. (2006) Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: Proceedings of the 8th international workshop on discrete event systems, WODES'06. IEEE, Ann Arbor, pp 384–385

Aloul FA, Markov IL, Sakallah KA (2003) FORCE: A fast & easy-to-implement variable-ordering heuristic. In: Proceedings of the 13th ACM great lakes symposium on VLSI, pp. 116–119. Washington, DC, USA

Baier C, Katoen JP (2008) Principles of model checking. MIT Press

Bryant RE (1992) Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput Surv 24(3):293–318

Chen Y, Lin F (2000) Modeling of discrete event systems using finite state machines with parameters. In: Proceedings of 2000 IEEE international conference on control applications (CCA). Anchorage, Alaska, pp 941–946

Cheng KT, Krishnakumar AS (1993) Automatic functional test generation using the extended finite state machine model. In: Proceedings of the 30th ACM/IEEE design automation conference, Dallas, pp 86–91. doi:10.1145/157485.164585

Dams D, Grumberg O, Gerth R (1994) Abstract interpretation of reactive systems: Abstractions preserving ∀CTL*, ∃CTL* and CTL*. In: Olderog ER (ed) Proceedings of IFIP WG2.1/WG2.2/WG2.3 working conference on programming concepts, methods and calculi (PROCOMET), IFIP transactions. Elsevier, Amsterdam, pp 573–592

Fabian M, Fei Z, Miremadi S, Lennartson B, Åkesson K (2014) Formal Methods in Manufacturing. In: Campos J, Seatzu C, Xie X (eds) Supervisory control of manufacturing systems using extended finite automata, CRC Press

Flordal H, Malik R (2009) Compositional verification in supervisory control. SIAM J Control Optim 48(3):1914–1938. doi:10.1137/070695526

Gohari P, Wonham WM (2000) On the complexity of supervisory control design in the RW framework. IEEE Trans Syst Man Cybern 30(5):643–652. doi:10.1109/3477.875441

Graf S, Steffen B (1990) Compositional minimization of finite state systems. In: Proceedings of the 1990 workshop on computer-aided verification, LNCS, vol 531. Springer, NJ, pp 186–196

Hoare CAR (1985) Communicating sequential processes. Prentice-Hall

Huth M, Ryan M (2004) Logic in computer science. University Press, Cambridge

Malik R, Leduc R (2013) Compositional nonblocking verification using generalised nonblocking abstractions. IEEE Trans Autom Control 58(8):1–13. doi:10.1109/TAC.2013.2248255

Malik R, Mühlfeld R. (2003) A case study in verification of UML statecharts: the PROFIsafe protocol. Journal of Universal Computer Science 9(2):138–151

Malik R, Streader D, Reeves S (2006) Conflicts and fair testing. Int J Found Comput Sci 17(4):797–813. doi:10.1142/S012905410600411X

McMillan KL (1993) Symbolic model checking. Kluwer Academic Publishers, Boston

Milner R (1989) Communication and concurrency. Series in Computer Science. Prentice-Hall

Mohajerani S, Malik R, Fabian M (2013) Compositional nonblocking verification for extended finite-state automata using partial unfolding. In: Proceedings of the 9th international conference on automation science and engineering, CASE. Wisconsin Press, Madison, pp 942–947

Mohajerani S, Malik R, Fabian M (2013) Partial unfolding for compositional nonblocking verification of extended finite-state machines. Working Paper 01/2013, Department of Computer Science, University of Waikato. Hamilton, New Zealand. http://hdl.handle.net/10289/7140

Mohajerani S, Malik R, Fabian M (2014) An algorithm for compositional nonblocking verification of extended finite-state machines. In: Proceedings of the 12th international workshop on discrete event systems, WODES'14, pp. 376–382. Paris, France

Parsaeian S (2014) Implementation of a framework for restart after unforeseen errors in manufacturing systems. Master's thesis, Chalmers University of Technology. Göteborg, Sweden

Pena PN, Cury JER, Lafortune S (2009) Verification of nonconflict of supervisors using abstractions. IEEE Trans Autom Control 54(12):2803–2815

Ramadge PJG, Wonham WM (1989) The control of discrete event systems. Proc IEEE 77(1):81–98

Sköldstam M, Åkesson K, Fabian M (2007) Modeling of discrete event systems using finite automata with variables. In: Proceedings of the 46th IEEE conference on decision and control, CDC '07, pp. 3387–3392

Su R, van Schuppen JH, Rooda JE, Hofkamp AT (2010) Nonconflict check by using sequential automaton abstractions based on weak observation equivalence. Automatica 46(6):968–978. doi:10.1016/j.automatica.2010.02.025

Teixeira M, Malik R, Cury JER, de Queiroz MH (2013) Variable abstraction and approximations in supervisory control synthesis. In: 2013 American Control Conference, pp. 120–125. Washington, DC, USA

Vahidi A (2004) Efficient analysis of discrete event systems—supervisor synthesis with binary decision diagrams. Ph.D. thesis. Chalmers University of Technology, Göteborg

Wonham WM Supervisory control of discrete-event systems. http://www.control.utoronto.edu/

Zhaoa J, Chen YL, Chen Z, Lin F, Wang C, Zhang H (2012) Modeling and control of discrete event systems using finite state machines with variables and their applications in power grids. Syst Control Lett 61(1):212–222

Sahar Mohajerani received her B.Sc. degree in Electrical Engineering from Khaje Nasir Toosi University of Technology, Tehran, Iran in 2005, her M.Sc. degree in Systems, Control, and Mechatronics in 2009, and her PhD degree in Automation in the Department of Signals and Systems from Chalmers University of Technology, Gothenburg, Sweden in 2015. She was awarded the best student paper award in 2014 from the 12th International Workshop on Discrete Event Systems. She is currently working at Volvo Cars Corporation on verification of active safety functions. Her research interests are in the area of formal methods for verification and synthesis of large discrete event systems.

Robi Malik received the M.S. and Ph.D. degree in computer science from the University of Kaiserslautern, Germany, in 1993 and 1997, respectively. From 1998 to 2002, he worked in a research and development group at Siemens Corporate Research in Munich, Germany, where he was involved in the development and application of modelling and analysis software for discrete event systems. Since 2003, he is lecturing at the Department of Computer Science at the University of Waikato in Hamilton, New Zealand. He is participating in the development of the Supremica software for modelling and analysis of discrete event systems. His current research interests are in the area of model checking and synthesis of large discrete event systems and other finite-state machine models.

Martin Fabian was born in Gothenburg, Sweden, in 1960. He received his Ph.D. degree in control engineering in 1995 from Chalmers University of Technology, Gteborg, Sweden. He is currently a Professor with the Department of Signals and Systems, Chalmers University of Technology. His research interests involve modelling and supervisory control of discrete-event systems, modular and compositional methods for complex systems, and generic architectures for flexible production systems.