Control Theory and Technology

Progressive events in supervisory control and compositional verification

Simon Ware [1] Robi Malik[†] [2]

1 *School of Electronic and Electrical Engineering, Nanyang Technological University, Singapore*

2 *Department of Computer Science, University of Waikato, Hamilton, New Zealand*

Received xxx; revised xxx; accepted xxx

Abstract:

This paper investigates some limitations of the *nonblocking* property when used for supervisor *synthesis* in discrete event systems. It is shown that there are cases where synthesis with the nonblocking property gives undesired results. To address such cases, the paper introduces *progressive events* as a means to specify more precisely how a synthesised supervisor should complete its tasks. The nonblocking property is modified to take progressive events into account, and appropriate methods for verification and synthesis are proposed. Experiments show that progressive events can be used in the analysis of industrial-scale systems, and can expose issues that remain undetected by standard nonblocking verification.

Keywords: Model validation in design methods; Controller constraints and structure; Computational issues

*DOI 10.1007/s11768-014-****-**

1 Introduction

In *supervisory control theory* [1,2], it is common to use the *nonblocking* property to ensure liveness when automatically synthesising supervisors. A discrete event system is nonblocking if, from every reachable state, all involved components can cooperatively complete their common tasks. It is not required that task completion is guaranteed on every possible execution path, only that there exists an execution path to a terminal state. For finite-state systems, the nonblocking property is equivalent to termination under the *fairness assumption* that events that are enabled infinitely often will be taken eventually [3]. This weak liveness condition ensures the existence of least restrictive synthesis results and has been used successfully in many applications [1,4].

On the other hand, the nonblocking property is weaker than a guarantee of termination, and it is not always expressive enough to give the intended results. Several alternatives and extensions to the standard nonblocking property have been proposed. *Multi-tasking supervisory control* [5] allows the specification of multiple nonblocking requirements that must be satisfied simultaneously. The *generalised nonblocking* property [6] restricts the situations in which nonblocking is required, which is useful in hierarchical interface-based supervisory control [7]. *Nonblocking under control* [8] changes the fairness assumption of standard nonblocking by making the assumption that controllable events can preempt uncontrollable events when completing tasks, facilitating reasoning about supervisor implementations. The authors of [9] replace the nonblocking property by the requirement of true termination and perform synthesis using ω-languages.

A different generalisation of the nonblocking property is proposed in [10]. Here, *progressive events* are introduced as the only events that can be used in traces towards task completion when checking the nonblocking property. Progressive events make it possible to capture nonblocking requirements in some cases where this is difficult with the standard nonblocking property, particularly when synthesis is involved, while verification verification and synthesis are still possible in the same computational complexity as with the standard nonblocking property. This paper is an extended version of [10]. It includes section 4.3 on compositional verification with some experimental results, which

[†]Corresponding author.

E-mail: robi@waikato.ac.nz; Tel.: +64 (0)7 838 4796; Fax: +64 (0)7 858 5095.

shows that progressive events can be used with industrial-scale discrete event systems, and that they can help to reveal issues that remain undetected by a standard nonblocking check.

In the following, Section 2 introduces the definitions for discrete event systems and supervisory control theory. Section 3 shows two examples of discrete event systems, for which the standard nonblocking property fails to give a useful synthesis result. Section 4 introduces progressive events to model these examples more appropriately, and shows how nonblocking verification and synthesis are adapted for progressive events. The section also includes a discussion of compositional verification methods, experimental results, and an algorithm for synthesis with progressive events. Afterwards, Section 5 compares nonblocking with progressive events to the other nonblocking properties mentioned above, and Section 6 adds some concluding remarks.

2 Preliminaries

2.1 Events and languages

The behaviour of discrete event systems is modelled using events and languages [1,2]. *Events* represent incidents that cause transitions from one state to another and are taken from a finite alphabet Σ. For the purpose of supervisory control, this alphabet is partitioned into the set $\Sigma_{\rm c}$ of *controllable* events and the set $\Sigma_{\rm uc}$ of *uncontrollable* events. Controllable events can be disabled by a supervising agent, while uncontrollable events cannot be disabled. Independently of this distinction, the alphabet Σ is also partitioned into the set $\Sigma_{\rm o}$ of *observable* events and the set $\Sigma_{\rm uo}$ of *unobservable* events. Observable events are visible to the supervising agent, while unobservable events are not. In this paper, it is assumed that all unobservable events are also uncontrollable.

Given an alphabet Σ, the term Σ^* denotes the set of all finite *traces* of the form $\sigma_1\sigma_2\cdots\sigma_n$ of events from Σ, including the *empty trace* ε. The *concatenation* of two traces $s, t \in \Sigma^*$ is written as st. A subset $L \subseteq \Sigma^*$ is called a *language*. A trace $s \in \Sigma^*$ is a *prefix* of $t \in \Sigma^*$, written $s \sqsubseteq t$, if $t = su$ for some $u \in \Sigma^*$. The *prefix-closure* of a language $L \subseteq \Sigma^*$ is $\overline{L} = \{\, s \in \Sigma^* \mid s \sqsubseteq t \text{ for some } t \in L \,\}$, and L is *prefix-closed* if $\overline{L} = L$.

For $\Omega \subseteq \Sigma$, the *natural projection* $P_{\Sigma\to\Omega}\colon \Sigma^* \to \Omega^*$ is the operation that removes from traces $s \in \Sigma^*$ all events not in Ω. Its inverse image $P^{-1}_{\Sigma\leftarrow\Omega}\colon \Omega^* \to 2^{\Sigma^*}$ is defined by $P^{-1}_{\Sigma\leftarrow\Omega}(t) = \{\, s \in \Sigma^* \mid P_{\Sigma\to\Omega}(s) = t \,\}$. If the source alphabet is clear from the context, these functions are also written as $P_\Omega = P_{\Sigma\to\Omega}$ and $P^{-1}_\Sigma = P^{-1}_{\Sigma\leftarrow\Omega}$.

The *synchronous composition* of two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ is $L_1 \parallel L_2 = P^{-1}_{\Sigma_1\cup\Sigma_2}(L_1) \cap P^{-1}_{\Sigma_1\cup\Sigma_2}(L_2)$.

2.2 Discrete event systems

In this paper, discrete event systems are modelled as pairs of languages or as finite-state automata.

Definition 1 Let Σ be a finite set of events. A *discrete event system* over Σ (Σ-*DES*) is a pair $\mathbf{L} = (L, L^\omega)$ where $L \subseteq \Sigma^*$ is a prefix-closed language, and $L^\omega \subseteq L$. These languages are also denoted by $\mathcal{L}(\mathbf{L}) = L$ and $\mathcal{L}^\omega(\mathbf{L}) = L^\omega$.

The prefix-closed behaviour $\mathcal{L}(\mathbf{L})$ contains possibly incomplete system executions. The (not necessarily prefix-closed) sublanguage $\mathcal{L}^\omega(\mathbf{L}) \subseteq \mathcal{L}(\mathbf{L})$ is the so-called *marked* behaviour and contains traces representing completed tasks.

Language operations are applied to discrete events systems by applying them to both components. For example, if $\mathbf{L}_i = (L_i, L_i^\omega)$ for $i = 1, 2$, then $\mathbf{L}_1 \parallel \mathbf{L}_2 = (L_1 \| L_2, L_1^\omega \| L_2^\omega)$, and the same notation is used for $\cup$. Discrete event systems form a lattice with inclusion, $\mathbf{L}_1 \subseteq \mathbf{L}_2$, defined to hold if and only if $L_1 \subseteq L_2$ and $L_1^\omega \subseteq L_2^\omega$.

Alternatively, it is common to model discrete event systems as finite-state machines or automata.

Definition 2 A (nondeterministic) *automaton* is a tuple $\mathbf{G} = \langle \Sigma, Q, \to, Q^\circ, Q^\omega \rangle$ where Σ is a finite set of *events*, Q is a set of *states*, $\to \subseteq Q \times \Sigma \times Q$ 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*.

$\mathbf{G}$ is *finite-state* if the state set Q is finite, and $\mathbf{G}$ is *deterministic* if $|Q^\circ| \le 1$ and $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$. Here, the transition relation is written in infix notation, $x \xrightarrow{\sigma} y$, and extended to traces in Σ^* in the standard way. Also, $\mathbf{G} \xrightarrow{s} x$ means $x^\circ \xrightarrow{s} x$ for some $x^\circ \in Q^\circ$. The prefix-closed and marked languages of an automaton $\mathbf{G}$ are

$$\mathcal{L}(\mathbf{G}) = \{\, s \in \Sigma^* \mid \mathbf{G} \xrightarrow{s} y \text{ for some } y \in Q \,\}\,; \tag{1}$$

$$\mathcal{L}^\omega(\mathbf{G}) = \{\, s \in \Sigma^* \mid \mathbf{G} \xrightarrow{s} y^\omega \text{ for some } y^\omega \in Q^\omega \,\}\,. \tag{2}$$

Using these definitions, an automaton $\mathbf{G}$ is also considered as the Σ-DES $\mathbf{G} = (\mathcal{L}(\mathbf{G}), \mathcal{L}^\omega(\mathbf{G}))$. Conversely, a discrete event system given by two languages is considered as an automaton by taking the canonical recogniser [11] of its languages.

2.3 Supervisory control

Given a *plant* $\mathbf{L}$ and a *specification* $\mathbf{K}$, *supervisory control theory* [1, 2] is concerned about the question whether and how the plant can be controlled in such a way that the specification is satisfied. This is dependent on the conditions of *controllability*, *normality*, and *nonblocking*.

Definition 3 Let $\mathbf{K}$ be a $\Sigma_{\mathbf{K}}$-DES, $\mathbf{L}$ a $\Sigma_{\mathbf{L}}$-DES, and let $\Sigma = \Sigma_{\mathbf{K}} \cup \Sigma_{\mathbf{L}}$. Then $\mathbf{K}$ is *controllable* with respect to $\mathbf{L}$ if

$$P_{\Sigma}^{-1}(\mathcal{L}(\mathbf{K}))\Sigma_{\mathrm{uc}} \cap P_{\Sigma}^{-1}(\mathcal{L}(\mathbf{L})) \subseteq P_{\Sigma}^{-1}(\mathcal{L}(\mathbf{K})) \ .$$

Definition 4 Let $\mathbf{K}$ be a $\Sigma_{\mathbf{K}}$-DES, $\mathbf{L}$ a $\Sigma_{\mathbf{L}}$-DES, and let $\Sigma = \Sigma_{\mathbf{K}} \cup \Sigma_{\mathbf{L}}$. Then $\mathbf{K}$ is *normal* with respect to $\mathbf{L}$ if

$$P_{\Sigma}^{-1}(P_{\Sigma_{\mathrm{o}} \cap \Sigma_{\mathbf{K}}}(\mathcal{L}(\mathbf{K}))) \cap P_{\Sigma}^{-1}(\mathcal{L}(\mathbf{L})) \subseteq P_{\Sigma}^{-1}(\mathcal{L}(\mathbf{K})) \ .$$

Controllability expresses that a supervisor cannot disable uncontrollable events, and normality expresses that a supervisor cannot detect the occurrence of unobservable events. Every controllable and normal behaviour can be implemented by a supervisor that only uses observable events as input and only disables controllable events.

In addition to the safety properties of controllability and normality, it is common to require the *nonblocking* property to ensure some form of liveness.

Definition 5 A Σ-DES $\mathbf{L}$ is called *standard nonblocking* (or simply *nonblocking*) if, for every trace $s \in \mathcal{L}(\mathbf{L})$, there exists a trace $t \in \Sigma^*$ such that $st \in \mathcal{L}^{\omega}(\mathbf{L})$.

If a given system behaviour $\mathbf{K}$ is not controllable, normal, or nonblocking, then this behaviour cannot be implemented through control or is undesirable due to *livelock* or *deadlock*. The question then arises whether $\mathbf{K}$ can somehow be modified to satisfy the requirements. A key result from supervisory control theory states that every DES $\mathbf{K}$ has a largest possible sub-behaviour $\mathbf{K}' \subseteq \mathbf{K}$ that exhibits the desired properties of controllability, normality, and nonblocking.

Theorem 1 [1] Let $\mathbf{K}$ and $\mathbf{L}$ be two DES. There exists a unique supremal sub-behaviour $\sup\mathcal{CN}(\mathbf{K}) \subseteq \mathbf{K}$ that is controllable, normal, and nonblocking:

$$\begin{aligned} \sup\mathcal{CN}_{\mathbf{L}}(\mathbf{K}) = \bigcup \{\, & \mathbf{K}' \subseteq \mathbf{K} \mid \mathbf{K}' \text{ is controllable} \\ & \text{and normal with respect to } \mathbf{L}, \\ & \text{and } \mathbf{K}' \parallel \mathbf{L} \text{ is nonblocking} \,\} \ . \end{aligned} \quad (3)$$

Furthermore, if $\mathbf{K}$ and $\mathbf{L}$ are represented by finite-state automata, a finite-state representation of the supremal controllable, normal, and nonblocking sub-behaviour $\sup\mathcal{CN}_{\mathbf{L}}(\mathbf{K})$ can be computed using a fixpoint iteration. This computation is called supervisor *synthesis*, and its result can be used to implement an appropriate supervisor [1].

3 Applications

This paper is concerned about the nonblocking property and its use in synthesis. In the following, two examples are discussed where the synthesis of a least restrictive supervisor using the standard nonblocking property from Definition 5 gives unexpected and probably undesirable results.

3.1 Computer-controlled board game

A board game is to be controlled, where a computer player and an opponent are taking moves in turn [6]. The control objective it to prevent the computer player from losing, while it is always possible for the game to end, either by the computer player winning or by a draw being declared. This is achieved by marking all states where the computer player has won, or the game is over without a winner. A least restrictive nonblocking supervisor can be synthesised to ensure that the game can always end in the desired way.

To complicate the example slightly, a reset feature is added: an additional event reset is introduced, which can always be executed by the environment and resets the game to its initial state. With this addition, the standard nonblocking property is much less expressive. Now, a least restrictive supervisor may allow the game to enter states where defeat for the computer player is inevitable, however due the omnipresent possibility of reset, the system is still nonblocking as long as there is some way of ending the game from its initial state. A synthesised supervisor may exploit this and make bad moves, knowing it is always possible to restart. In this modified model, it is much more interesting to synthesise a supervisor to ensure that *"the game can always end, even if* reset *is not used."*

3.2 Manufacturing cell

Fig. 1 shows a modified version of a manufacturing cell proposed in [12], which consists of a robot, a machine, two conveyors, two buffers, and a switch. The machine (plant $\mathbf{machine}$) can manufacture two types of products. Event $\mathsf{start}[k]$ initiates the manufacturing of a type k product ($k = 1$ or $k = 2$) from a workpiece in input buffer $\mathbf{inbuf}$, which upon completion is placed in output buffer $\mathbf{outbuf}$, indicated by the uncontrollable event $!\mathsf{finish}[k]$. The robot

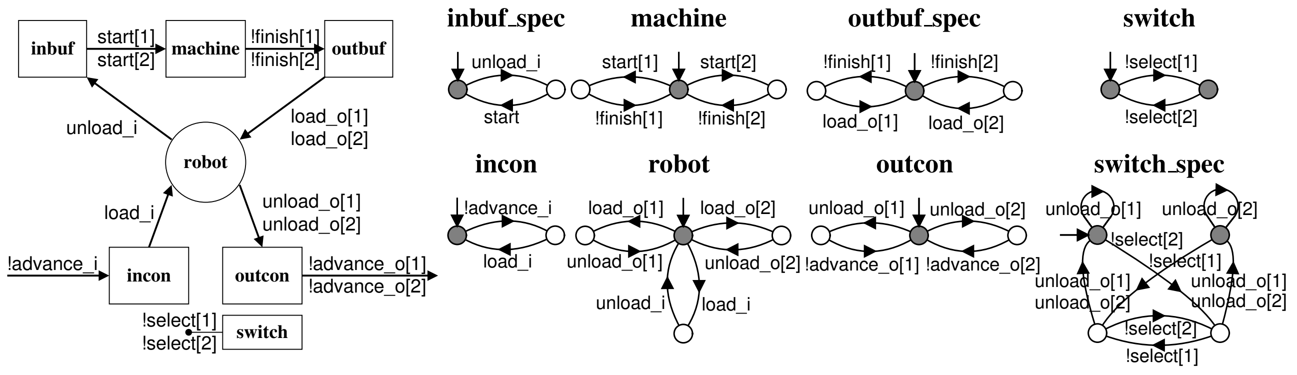

Fig. 1 Manufacturing cell example. Uncontrollable events are prefixed with !, and all events are observable.

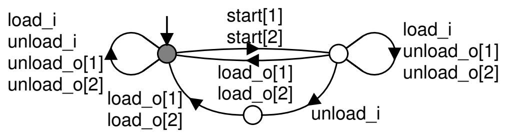

Fig. 2 Synthesised manufacturing cell supervisor.

(plant $\mathbf{robot}$) takes workpieces from the input conveyor (plant $\mathbf{incon}$) on event load_i and puts them in $\mathbf{inbuf}$ on event unload_i, and it takes type k products from $\mathbf{outbuf}$ on event load_o$[k]$ and puts them on the output conveyor (plant $\mathbf{outcon}$) on event unload_o$[k]$. The conveyors can be advanced to bring in new workpieces (!advance_i), or to remove completed products (!advance_o$[k]$). Specifications $\mathbf{inbuf\_spec}$ and $\mathbf{outbuf\_spec}$ request a supervisor that prevents overflow and underflow of two one-place buffers.

In addition, there is a switch (plant $\mathbf{switch}$) that allows the user to choose the type of products to be delivered. Specification $\mathbf{switch\_spec}$ requires that, when the user changes the desired output type to k (!select$[k]$), at most one product of the other type may be released from the cell; after that only type k products may be released (unload_o$[k]$) until the switch is operated again.

The model in Fig. 1 is not controllable and blocking. Standard synthesis [1] with supervisor reduction [13] gives the least restrictive supervisor in Fig. 2. This supervisor correctly prevents buffer overflow by not allowing the machine to start before the output buffer is empty, and prevents deadlock by restricting the number of workpieces in the cell to two.

The supervisor does not distinguish start$[1]$ and start$[2]$, always allowing both types of products to be manufactured. This works because specification $\mathbf{switch\_spec}$ can be satisfied by disabling the controllable event unload_o$[k]$ when the robot holds a workpiece of an undesired type k, delaying delivery until the user changes the switch with another !select$[k]$ event. While this is the least restrictive controllable and nonblocking behaviour, it seems unreasonable to delay delivery and override the user's choice in this way. A more reasonable supervisor would respect the user's choice when starting the machine, instead of relying on the user to request delivery of what has already been produced.

4 Nonblocking with progressive events

4.1 Progressive events

To provide a better way of modelling examples such as those in Section 3, this section proposes to distinguish events that can be used to establish the nonblocking property from other events. Independently of controllability and observability, the event set Σ is partitioned into the sets $\Sigma_{\rm p}$ of *progressive* events and $\Sigma_{\rm np}$ of *non-progressive* events.

Definition 6 Let $\mathbf{L}$ be a Σ-DES, and let $\Sigma_{\rm p} \subseteq \Sigma$. Then $\mathbf{L}$ is $\Sigma_{\rm p}$-*nonblocking* if, for every trace $s \in \mathcal{L}(\mathbf{L})$, there exists a trace $t \in \Sigma_{\rm p}^*$ such that $st \in \mathcal{L}^{\omega}(\mathbf{L})$.

Nonblocking with progressive events requires that, from all reachable states, it is possible to reach a marked state using only progressive events. Non-progressive events are assumed to occur only occasionally or as external input, and a supervisor should not rely on them for task completion.

Definition 7 Let $\mathbf{K}$ and $\mathbf{L}$ be two DES, and let $\Sigma_{\rm p}$ be a set of progressive events. The least restrictive controllable, normal, and $\Sigma_{\rm p}$-nonblocking sub-behaviour of $\mathbf{K}$ with respect to $\mathbf{L}$ is

$$\sup\mathcal{CN}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}) = \bigcup \{\, \mathbf{K}' \subseteq \mathbf{K} \mid \mathbf{K}' \text{ is controllable and normal with respect to } \mathbf{L}, \text{ and } \mathbf{K}' \parallel \mathbf{L} \text{ is } \Sigma_{\rm p}\text{-nonblocking} \,\}\,.$$

Definition 7 redefines the objective of synthesis to use

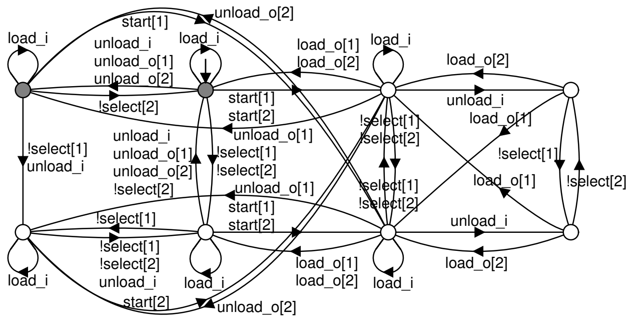

Fig. 3 Synthesised manufacturing cell supervisor with progressive events.

the modified nonblocking property. It follows from Proposition 2 below that the definition is sound in that it indeed defines a controllable, normal, and $\Sigma_{\rm p}$-nonblocking behaviour.

In Section 3, events reset and !select$[k]$ would be non-progressive. Then a $\Sigma_{\rm p}$-nonblocking supervisor ensures task completion even if the game is not reset, or the manufacturing cell user never changes the requested workpiece type. Fig. 3 shows a least restrictive reduced supervisor for the manufacturing cell subject to the !select$[k]$ events being non-progressive. In addition to preventing buffer overflow and deadlock, this supervisor prevents the machine from producing a second workpiece while another is being delivered.

4.2 Relationship to standard nonblocking

This section relates the nonblocking property with progressive events to the standard nonblocking property. As Definitions 5 and 6 coincide when $\Sigma_{\rm p} = \Sigma$, it is clear that standard nonblocking is a special case of nonblocking with progressive events. If there are non-progressive events, then nonblocking with progressive events is a stronger condition.

Yet, nonblocking with progressive events can be expressed using standard nonblocking by means of an additional DES $\mathbf{P}(\Sigma_{\rm np}, \tau)$ as shown in Fig. 4, which uses a new event τ that disables all non-progressive events. Initially, non-progressive events are possible, but τ may be executed at any time, taking $\mathbf{P}(\Sigma_{\rm np}, \tau)$ to state p_1 where only progressive events can occur. When $\mathbf{P}(\Sigma_{\rm np}, \tau)$ is composed with a system to be analysed, all states remain reachable, yet standard nonblocking can only hold if marked states can be reached using progressive events only.

Definition 8 Let $\Sigma_{\rm np}$ be a set of events. The *progressive DES* $\mathbf{P}(\Sigma_{\rm np}, \tau) = (\overline{\Sigma_{\rm np}^*\tau}, \Sigma_{\rm np}^*\tau)$ for $\Sigma_{\rm np}$ and $\tau \notin \Sigma_{\rm np}$ is the $(\Sigma_{\rm np} \cup \{\tau\})$-DES shown in Fig. 4.

Proposition 2 Let $\mathbf{L}$ be a Σ-DES with $\Sigma = \Sigma_{\rm p} \dot\cup \Sigma_{\rm np}$

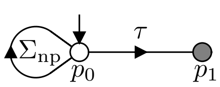

Fig. 4 The DES $\mathbf{P}(\Sigma_{\rm np}, \tau)$ to express $\Sigma_{\rm p}$-nonblocking as standard nonblocking. The selfloop marked $\Sigma_{\rm np}$ stands for transitions with all events in $\Sigma_{\rm np}$, and $\tau \notin \Sigma$ is a new event that does not appear elsewhere in the system.

and $\tau \notin \Sigma$. Then $\mathbf{L}$ is $\Sigma_{\rm p}$-nonblocking if and only if $\mathbf{L} \parallel \mathbf{P}(\Sigma_{\rm np}, \tau)$ is standard nonblocking.

Proof Let $\mathbf{P} = \mathbf{P}(\Sigma_{\rm np}, \tau)$.

First assume that $\mathbf{L}$ is $\Sigma_{\rm p}$-nonblocking, and let $s \in \mathcal{L}(\mathbf{L} \parallel \mathbf{P})$. Then first note that $P_{\Sigma_{\rm np}\cup\{\tau\}}(s) \in \mathcal{L}(\mathbf{P}) = \overline{\Sigma_{\rm np}^*\tau}$. Let $t = \tau$ if the event τ does not appear in s, and let $t = \varepsilon$ if τ appears in s. It follows that $P_{\Sigma_{\rm np}\cup\{\tau\}}(st) \in \Sigma_{\rm np}^*\tau = \mathcal{L}^\omega(\mathbf{P})$ and thus $st \in P_{\Sigma\cup\{\tau\}}^{-1}(\mathcal{L}^\omega(\mathbf{P}))$. Furthermore, note that $P_\Sigma(st) = P_\Sigma(s) \in \mathcal{L}(\mathbf{L})$, and as $\mathbf{L}$ is $\Sigma_{\rm p}$-nonblocking, there exists $u \in \Sigma_{\rm p}^*$ such that $P_\Sigma(st)u \in \mathcal{L}^\omega(\mathbf{L})$, This implies $stu \in P_{\Sigma\cup\{\tau\}}^{-1}(\mathcal{L}^\omega(\mathbf{L}))$. Also since $u \in \Sigma_{\rm p}^*$, it holds that $P_{\Sigma_{\rm np}\cup\{\tau\}}(stu) = P_{\Sigma_{\rm np}\cup\{\tau\}}(st) \in \mathcal{L}^\omega(\mathbf{P})$, and thus $stu \in P_{\Sigma\cup\{\tau\}}^{-1}(\mathcal{L}^\omega(\mathbf{P}))$. Therefore $stu \in \mathcal{L}^\omega(\mathbf{L} \parallel \mathbf{P})$, i.e., $\mathbf{L} \parallel \mathbf{P}$ is standard nonblocking.

Conversely assume $\mathbf{L} \parallel \mathbf{P}$ is standard nonblocking, and let $s \in \mathcal{L}(\mathbf{L})$. Then $s\tau \in \mathcal{L}(\mathbf{L} \parallel \mathbf{P})$. As $\mathbf{L} \parallel \mathbf{P}$ is nonblocking, there exists $u \in \Sigma^*$ such that $s\tau u \in \mathcal{L}^\omega(\mathbf{L} \parallel \mathbf{P})$. Then $P_{\Sigma_{\rm np}\cup\{\tau\}}(s\tau u) \in \mathcal{L}^\omega(\mathbf{P})$, which by construction of $\mathbf{P}$ implies $P_{\Sigma_{\rm np}\cup\{\tau\}}(u) = \varepsilon$, i.e., $u \in \Sigma_{\rm p}^*$. Since furthermore $su = P_\Sigma(s\tau u) \in \mathcal{L}^\omega(\mathbf{L})$, it follows that $\mathbf{L}$ is $\Sigma_{\rm p}$-nonblocking. □

Proposition 2 shows that any nonblocking verification task with progressive events can be reduced to a standard nonblocking verification task. However, composition with the progressive automaton $\mathbf{P}(\Sigma_{\rm np}, \tau)$ doubles the state space and verification time.

The extra effort is not necessary. Standard nonblocking can be checked by searching backwards from marked states to see whether all states are reached. By changing the backward search to use progressive events only, nonblocking with progressive events can be checked on the original system state space, exploring less transitions than a standard nonblocking check.

Proposition 2 is of theoretical interest, because it shows that progressive events do not add to the expressive power of standard nonblocking, and it can be of practical use, because it shows that a wide variety of nonblocking verification algorithms, particularly *compositional* verification, can also be used with progressive events. This is explained in detail in Section 4.3 below.

It is not immediately clear whether the progressive DES $\mathbf{P}(\Sigma_{\rm np}, \tau)$ can also be used to express synthesis with progressive events as standard synthesis. Indeed, if there are uncontrollable non-progressive events, then $\mathbf{P}(\Sigma_{\rm np}, \tau)$ used as an additional plant will disable some uncontrollable events, and a supervisor could wait for the auxiliary event τ to occur in order to avoid controllability problems.

This is avoided if τ is *unobservable*. Then the supervisor cannot distinguish the states of $\mathbf{P}(\Sigma_{\rm np}, \tau)$, so it has to enable uncontrollable events enabled in p_0 and at the same time ensure task completion from p_1. Lemma 3 shows for unobservable τ that controllability and normality are preserved by the addition of $\mathbf{P}(\Sigma_{\rm np}, \tau)$, which together with Proposition 2 implies the preservation of synthesis results as shown in Proposition 4.

Lemma 3 Let $\mathbf{K}$ and $\mathbf{L}$ be Σ-DES with $\Sigma = \Sigma_{\rm p} \dot\cup \Sigma_{\rm np}$, and let $\tau \notin \Sigma$ be an uncontrollable and unobservable event.

(i) $\mathbf{K}$ is controllable with respect to $\mathbf{L}$ if and only if $\mathbf{K}$ is controllable with respect to $\mathbf{L} \parallel \mathbf{P}(\Sigma_{\rm np}, \tau)$.

(ii) $\mathbf{K}$ is normal with respect to $\mathbf{L}$ if and only if $\mathbf{K}$ is normal with respect to $\mathbf{L} \parallel \mathbf{P}(\Sigma_{\rm np}, \tau)$.

Proof Let $\mathbf{P} = \mathbf{P}(\Sigma_{\rm np}, \tau)$.

(i) First assume that $\mathbf{K}$ is controllable with respect to $\mathbf{L} \parallel \mathbf{P}$, and let $sv \in \mathcal{L}(\mathbf{K})\Sigma_{\rm uc} \cap \mathcal{L}(\mathbf{L})$. Then $sv \in \Sigma^*$, and $P_{\Sigma_{\rm np}\cup\{\tau\}}(sv) \in \Sigma_{\rm np}^* \subseteq \mathcal{L}(\mathbf{P})$ by construction of $\mathbf{P}$, and thus $sv \in P^{-1}_{\Sigma\cup\{\tau\}}(\mathcal{L}(\mathbf{K}))\Sigma_{\rm uc} \cap \mathcal{L}(\mathbf{L} \parallel \mathbf{P}) \subseteq P^{-1}_{\Sigma\cup\{\tau\}}(\mathcal{L}(\mathbf{K}))$ as $\mathbf{K}$ is controllable with respect to $\mathbf{L} \parallel \mathbf{P}$. It follows that $sv \in \mathcal{L}(\mathbf{K})$, which means that $\mathbf{K}$ is controllable with respect to $\mathbf{L}$. The converse inclusion holds by Proposition 3 in [14].

(ii) First assume that $\mathbf{K}$ is normal with respect to $\mathbf{L}$, and let $s \in P^{-1}_{\Sigma\cup\{\tau\}}(P_{\Sigma_{\rm o}}(\mathcal{L}(\mathbf{K}))) \cap \mathcal{L}(\mathbf{L} \parallel \mathbf{P})$. Then clearly $P_\Sigma(s) \in P_\Sigma^{-1}(P_{\Sigma_{\rm o}}(\mathcal{L}(\mathbf{K}))) \cap \mathcal{L}(\mathbf{L}) \subseteq \mathcal{L}(\mathbf{K})$ as $\mathbf{K}$ is normal with respect to $\mathbf{L}$. Thus, $\mathbf{K}$ is normal with respect to $\mathbf{L} \parallel \mathbf{P}$.

Conversely assume $\mathbf{K}$ is normal with respect to $\mathbf{L} \parallel \mathbf{P}$, and let $s \in P_\Sigma^{-1}(P_{\Sigma_{\rm o}}(\mathcal{L}(\mathbf{K}))) \cap \mathcal{L}(\mathbf{L})$. Then $s \in \Sigma^*$ and therefore $P_{\Sigma_{\rm np}\cup\{\tau\}}(s) \in \Sigma_{\rm np}^* \subseteq \mathcal{L}(\mathbf{P})$, which implies $s \in P^{-1}_{\Sigma\cup\{\tau\}}(P_{\Sigma_{\rm o}}(\mathcal{L}(\mathbf{K}))) \cap \mathcal{L}(\mathbf{L} \parallel \mathbf{P}) \subseteq \mathcal{L}(\mathbf{K})$ as $\mathbf{K}$ is normal with respect to $\mathbf{L} \parallel \mathbf{P}$. This shows that $\mathbf{K}$ is normal with respect to $\mathbf{L}$. □

Proposition 4 Let $\mathbf{K}$ and $\mathbf{L}$ be Σ-DES with $\Sigma = \Sigma_{\rm p} \dot\cup \Sigma_{\rm np}$, and let $\tau \notin \Sigma$ be an uncontrollable and unobservable event. Then

$$\sup\mathcal{CN}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}) = P_\Sigma(\sup\mathcal{CN}_{\mathbf{L}\parallel\mathbf{P}(\Sigma_{\rm np},\tau)}(\mathbf{K}))\ . \tag{4}$$

Proof Consider an arbitrary sub-behaviour $\mathbf{K}' \subseteq \mathbf{K}$. In Lemma 3 it has been shown that $\mathbf{K}'$ is controllable and normal with respect to $\mathbf{L}$ if and only if $\mathbf{K}'$ is controllable and normal with respect to $\mathbf{L} \parallel \mathbf{P}(\Sigma_{\rm np}, \tau)$, and in Proposition 2 it has been shown that $\mathbf{K}' \parallel \mathbf{L}$ is $\Sigma_{\rm p}$-nonblocking if and only if $\mathbf{K}' \parallel \mathbf{L} \parallel \mathbf{P}(\Sigma_{\rm np}, \tau)$ is nonblocking. As this holds for all sub-behaviours $\mathbf{K}'$ of $\mathbf{K}$, the least restrictive sub-behaviours must also be equal. □

Thus, synthesis with progressive events can be achieved using standard synthesis methods. However, the introduced automaton $\mathbf{P}(\Sigma_{\rm np}, \tau)$ includes the unobservable event τ, making it necessary to use the more complex synthesis algorithm with unobservable events [2], even if the original model only has observable events. Section 4.4 below presents a direct algorithm for synthesis with progressive events that does not have these performance issues.

4.3 Compositional verification

This section investigates compositional verification and shows how the nonblocking property with progressive events can be verified efficiently for large systems.

The standard method to check whether a system is nonblocking [2] involves the explicit composition of all the automata involved, and is limited by the well-known *state-space explosion* problem. *Compositional verification* [15, 16] is an effective alternative that works by simplifying individual automata of a large synchronous composition, gradually reducing the state space of the system and allowing much larger systems to be verified in the end. Compositional verification requires the use of abstraction methods that preserve the property being verified.

While no abstraction methods have been developed for nonblocking with progressive events, Proposition 2 shows that a nonblocking check with progressive events can be replaced by a standard nonblocking check after the addition of a single automaton $\mathbf{P}(\Sigma_{\rm np}, \tau)$. This makes it possible to apply all the techniques that exist for compositional verification of the standard nonblocking property [17–20]. These techniques are based on the preservation of *conflict equivalence*, which is the most general process equivalence for use in compositional nonblocking verification [21]. If a component of a system is replaced by a conflict equivalent component, the nonblocking property is guaranteed to be preserved.

Compositional algorithms verify whether a set $\mathbf{G}$ of automata is nonblocking by taking a subset $\mathbf{H} \subseteq \mathbf{G}$ of the automata and composing them to create an automaton $H = \parallel\mathbf{H}$. Then the set of *local* events of H is identified: these are events that appear only in $\mathbf{H}$ and not in the rest of the system $\mathbf{G} \setminus \mathbf{H}$. The local events are *hidden* from H, i.e., they are replaced by a new event $\tau_{\mathbf{H}} \notin \Sigma$, resulting in a new automaton H'. Then *abstraction* techniques [17–20] are used to simplify H' and obtain a conflict equivalent abstraction H''. Because H'' is conflict equivalent to H', and H' is obtained by hiding local events from H, it can be shown [21] that H synchronised with the automata in $\mathbf{G} \setminus \mathbf{H}$ is nonblocking if and only if H'' composed with the same automata is nonblocking. Therefore, the problem to verify whether the set of automata $\mathbf{G}$ is nonblocking is replaced by the equivalent problem to verify whether the simpler set of automata $(\mathbf{G} \setminus \mathbf{H}) \cup \{H'\}$ is nonblocking. This procedure is repeated until the set of automata is simple enough to be composed together in a standard nonblocking check.

The above algorithm relies on local events. Thus the addition of a single progressive automaton $\mathbf{P}(\Sigma_{\mathrm{np}}, \tau)$ can be problematic, because it increases the coupling between these events in the model. If there are a lot of non-progressive events that are used by a lot of automata, then many automata may have to be composed with $\mathbf{P}(\Sigma_{\mathrm{np}}, \tau)$ before events can be removed. The following Proposition 5 suggests a way to avoid this problem by splitting the progressive automaton $\mathbf{P}(\Sigma_{\mathrm{np}}, \tau)$ into smaller automata. It is possible to create separate automata $\mathbf{P}(\Sigma^i_{\mathrm{np}}, \tau^i)$ for different subsets Σ^i_{np} of the set of non-progressive events. The proposition shows that, no matter what the system $\mathbf{T}$ to be verified is, $\mathbf{T} \parallel \mathbf{P}(\Sigma_{\mathrm{np}}, \tau)$ is nonblocking if and only if $\mathbf{T} \parallel \big\|_i \mathbf{P}(\Sigma^i_{\mathrm{np}}, \tau^i)$ is nonblocking.

Proposition 5 Let $\Sigma_1, \Sigma_2 \subseteq \Sigma$ be sets of events, and let $\tau, \tau_1, \tau_2 \notin \Sigma$ be three distinct events. For every Σ-DES $\mathbf{T}$, it holds that $\mathbf{T} \parallel \mathbf{P}(\Sigma_1 \cup \Sigma_2, \tau)$ is nonblocking if and only if $\mathbf{T} \parallel \mathbf{P}(\Sigma_1, \tau_1) \parallel \mathbf{P}(\Sigma_2, \tau_2)$ is nonblocking.

Proof Let $\Sigma_{12} = \Sigma_1 \cup \Sigma_2$, $\mathbf{P}_{12} = \mathbf{P}(\Sigma_{12}, \tau)$, $\mathbf{P}_1 = \mathbf{P}(\Sigma_1, \tau_1)$, and $\mathbf{P}_2 = \mathbf{P}(\Sigma_2, \tau_2)$. Then it is to be shown that $\mathbf{T} \parallel \mathbf{P}_{12}$ is nonblocking if and only if $\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2$ is nonblocking.

First assume that $\mathbf{T} \parallel \mathbf{P}_{12}$ is nonblocking, and let $s \in \mathcal{L}(\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2)$. For $i = 1, 2$, let $t_i = \tau_i$ if the event τ_i does not appear in s, and $t_i = \varepsilon$ if τ_i appears in s. Then $P_{\Sigma_i \cup \{\tau_i\}}(st_i) \in \mathcal{L}^\omega(\mathbf{P}_i)$ for $i = 1, 2$ by Definition 8, and $st_1t_2 \in \mathcal{L}(\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2)$. Furthermore, $P_\Sigma(s) \in \mathcal{L}(\mathbf{T} \parallel \mathbf{P}_{12})$ as $P_{\Sigma_{12}}(s) \in \Sigma^*_{12} \subseteq \overline{\Sigma^*_{12}\tau} = \mathcal{L}(\mathbf{P}(\Sigma_{12}, \tau)) = \mathcal{L}(\mathbf{P}_{12})$ by Definition 8. As $\tau \notin \Sigma$, it follows that $P_\Sigma(s)\tau \in \mathcal{L}(\mathbf{T} \parallel \mathbf{P}_{12})$. As $\mathbf{T} \parallel \mathbf{P}_{12}$ is nonblocking, there exists a trace $u \in (\Sigma \cup \{\tau\})^*$ such that $P_\Sigma(s)\tau u \in \mathcal{L}^\omega(\mathbf{T} \parallel \mathbf{P}_{12})$. By Definition 8, it follows that $P_{\Sigma_{12} \cup \{\tau\}}(s)\tau P_{\Sigma_{12} \cup \{\tau\}}(u) = P_{\Sigma_{12} \cup \{\tau\}}(P_\Sigma(s)\tau u) \in \mathcal{L}^\omega(\mathbf{P}_{12}) = \Sigma^*_{12}\tau$ and thus $u \in (\Sigma \setminus \Sigma_{12})^*$, and as $\tau, \tau_1, \tau_2 \notin \Sigma$ it holds that $P_\Sigma(st_1t_2u) = P_\Sigma(su) = P_\Sigma(P_\Sigma(s)\tau u) \in \mathcal{L}^\omega(\mathbf{T})$. As $u \in (\Sigma \setminus \Sigma_{12})^*$, it holds that $P_{\Sigma_i \cup \{\tau_i\}}(u) = \varepsilon$ and thus $P_{\Sigma_i \cup \{\tau_i\}}(st_1t_2u) = P_{\Sigma_i \cup \{\tau_i\}}(st_i) \in \mathcal{L}^\omega(\mathbf{P}_i)$ for $i = 1, 2$. Hence $st_1t_2u \in \mathcal{L}^\omega(\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2)$, i.e., $\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2$ is nonblocking.

Now assume that $\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2$ is nonblocking, and let $s \in \mathcal{L}(\mathbf{T} \parallel \mathbf{P}_{12})$. Let $t = \tau$ if the event τ does not appear in s, and $t = \varepsilon$ if τ appears in s. Then $P_{\Sigma \cup \{\tau\}}(st) \in \mathcal{L}^\omega(\mathbf{P}_{12})$ by Definition 8, and $st \in \mathcal{L}(\mathbf{T} \parallel \mathbf{P}_{12})$. Furthermore, $P_\Sigma(s) \in \mathcal{L}(\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2)$, as $P_{\Sigma_i \cup \{\tau_i\}}(P_\Sigma(s)) = P_{\Sigma_i}(s) \in \Sigma^*_i \subseteq \overline{\Sigma^*_i\tau_i} = \mathcal{L}(\mathbf{P}_i)$ for $i = 1, 2$ by Definition 8. As $\tau_1, \tau_2 \notin \Sigma$ it holds that $P_\Sigma(s)\tau_1\tau_2 \in \mathcal{L}(\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2)$. As $\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2$ is nonblocking, there exists a trace $u \in (\Sigma_{12} \cup \{\tau_1, \tau_2\})^*$ such that $P_\Sigma(s)\tau_1\tau_2u \in \mathcal{L}^\omega(\mathbf{T} \parallel \mathbf{P}_1 \parallel \mathbf{P}_2)$. By Definition 8, it follows that $P_{\Sigma_i \cup \{\tau_i\}}(s)\tau_i P_{\Sigma_i \cup \{\tau_i\}}(u) = P_{\Sigma_i \cup \{\tau_i\}}(P_\Sigma(s)\tau_1\tau_2u) \in \mathcal{L}^\omega(\mathbf{P}_i) = \Sigma^*_i\tau_i$ for $i = 1, 2$ and thus $u \in (\Sigma \setminus \Sigma_{12})^*$, and as $\tau, \tau_1, \tau_2 \notin \Sigma$ it holds that $P_\Sigma(stu) = P_\Sigma(su) = P_\Sigma(P_\Sigma(s)\tau_1\tau_2u) \in \mathcal{L}^\omega(\mathbf{T})$. As $u \in (\Sigma \setminus \Sigma_{12})^*$, it holds that $P_{\Sigma_{12} \cup \{\tau\}}(u) = \varepsilon$ and thus $P_{\Sigma_{12} \cup \{\tau\}}(stu) = P_{\Sigma_{12} \cup \{\tau\}}(st) \in \mathcal{L}^\omega(\mathbf{P}_{12})$. Hence $stu \in \mathcal{L}^\omega(\mathbf{T} \parallel \mathbf{P}_{12})$, i.e., $\mathbf{T} \parallel \mathbf{P}_{12}$ is nonblocking. □

The compositional nonblocking checker implemented in the DES software tool Supremica [22] has been used to check the nonblocking property of five discrete event systems. One of these is the example given in Section 3.2 above, while the other four are industrial-scale models also used as benchmarks for compositional verification in [23], where a reasonable set of non-progressive events was identified. The following list gives some more information about these models.

aip0aip Model of the automated manufacturing system of the Atelier Inter-établissement de Productique [24]. Considered here is an early version based on [25].

big_bmw BMW window lift controller model from Petra Malik's dissertation [26].

cell_switch Manufacturing cell model described in Section 3.2. The model considered for the experiments consists of the automata in Fig. 1 and the supervisor in Fig. 2, and is Σ_{p}-blocking.

tip3 Model of the interaction between a mobile client and event-based servers of a tourist information system [27].

verriegel4 Car central locking system, originally from the KORSYS project [28].

Table 1 shows the results of compositional verification of the nonblocking property with progressive events for the above models. The "Size" column refers to the total number of states in the full synchronous composition of each model, without the additional progressive events automata, and the "Result" column indicates whether or not the model is nonblocking with progressive events. The columns "Single **P**" and "Multiple **P**" refer to two ways of performing the compositional nonblocking check. In the case of "Single **P**", only one progressive automaton is created for all non-progressive events, whereas in the case

Table 1 Experimental results

Model	Size	Result	Single $\mathbf{P}$		Multiple $\mathbf{P}$	
			Peak States	Time [s]	Peak States	Time [s]
aip0aip	$1.0 \cdot 10^8$	true	$392,767$	78.284	$45,740$	38.463
big_bmw	$3.1 \cdot 10^7$	true	$1,532$	2.751	$2,847$	4.464
cell_switch	672	false	118	0.190	148	0.186
tip3	$2.3 \cdot 10^{11}$	true	$184,238$	211.3		
verriegel4	$4.5 \cdot 10^{10}$	false	327	0.192	$1,261,250$	539.8

of "Multiple $\mathbf{P}$", separate progressive automata are used, each containing the non-progressive events of a single system component. For each experiment, the "Peak States" column shows the number of states of the largest automaton constructed during the check, and "Time" is the number of seconds taken to complete the check. The entries for the **tip3** model with the "Multiple $\mathbf{P}$" method are blank, because the algorithm ran out of memory in this case.

The results show that compositional nonblocking verification works well to check the nonblocking property with progressive events of large models. In most cases, using only one progressive events automaton works better than splitting it, with the exception of the **aip0aip** model. This may be because a larger number of automata means more work, also for compositional algorithms, or because the compositional algorithms has no knowledge about the progressive events automata and may compose them with other automata than the ones they were created for. It is possible that performance can be improved using a more specific composition strategy.

Verification of the central locking system model **verriegel4** shows that it is blocking with progressive events, although it is standard nonblocking. This is an unexpected result, and investigation of the counterexamples suggests an issue with the controller in that it exhibits a deadlock-like situation after two simultaneous requests to unlock the car, which can only be resolved after the arrival of another request. This suspected controller bug was not found by the standard nonblocking checks performed on the model before.

4.4 Direct synthesis algorithm

This section proposes a direct synthesis algorithm with progressive events for the case of *total observation*, i.e., when all events are observable. In this case, the unobservable event τ can be avoided, which gives rise to a more efficient solution. The following synthesis objective is considered.

Definition 9 Let $\mathbf{K}$ and $\mathbf{L}$ be Σ-DES, and let $\Sigma_{\rm p} \subseteq \Sigma$. The least restrictive controllable and $\Sigma_{\rm p}$-nonblocking sub-behaviour of $\mathbf{K}$ with respect to $\mathbf{L}$ is

$$\sup\mathcal{C}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}) = \bigcup \{\, \mathbf{K}' \subseteq \mathbf{K} \mid \mathbf{K}' \text{ is controllable with respect to } \mathbf{L}\text{, and } \mathbf{K}' \parallel \mathbf{L} \text{ is } \Sigma_{\rm p}\text{-nonblocking} \,\}\,. \tag{5}$$

The following Definition 10 defines a synthesis operator on the sub-behaviours of $\mathbf{L}$, which afterwards is shown to have the above $\sup\mathcal{C}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$ as its *greatest fixpoint* [29].

Definition 10 Let $\mathbf{L}$ be a Σ-DES, and let $\Sigma_{\rm p} \subseteq \Sigma$. The operator $\Theta_{\mathbf{L},\Sigma_{\rm p}}$ on the lattice of Σ-DES is defined by

$$\Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}) = (\theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}), \theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}) \cap \mathcal{L}^{\omega}(\mathbf{K}))\,; \tag{6}$$

$$\theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}) = \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}) \cap \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})\,; \tag{7}$$

$$\theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}) = \{\, s \in \mathcal{L}(\mathbf{K}) \mid \text{for all } r \sqsubseteq s \text{ and } \upsilon \in \Sigma_{\rm uc} \text{ such that } r\upsilon \in \mathcal{L}(\mathbf{L})\text{, it holds that } r\upsilon \in \mathcal{L}(\mathbf{K}) \,\}\,;$$

$$\theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}) = \{\, s \in \mathcal{L}(\mathbf{K}) \mid \text{for all } r \sqsubseteq s \text{ there exists } t \in \Sigma_{\rm p}^* \text{ such that } rt \in \mathcal{L}^{\omega}(\mathbf{L} \parallel \mathbf{K}) \,\}\,.$$

It is first shown that the post-fixpoints of $\Theta_{\mathbf{L},\Sigma_{\rm p}}$ are exactly the controllable and $\Sigma_{\rm p}$-nonblocking sub-behaviours of $\mathbf{L}$.

Proposition 6 Let $\mathbf{L}$ and $\mathbf{K}$ be a Σ-DES such that $\mathbf{K} \subseteq \mathbf{L}$, and let $\Sigma_{\rm p} \subseteq \Sigma$. Then $\mathbf{K} \subseteq \Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$, if and only if $\mathbf{K}$ is controllable with respect to $\mathbf{L}$ and $\mathbf{L} \parallel \mathbf{K}$ is $\Sigma_{\rm p}$-nonblocking.

Proof First assume $\mathbf{K} \subseteq \Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$. To see that $\mathbf{K}$ is controllable with respect to $\mathbf{L}$, let $s \in \mathcal{L}(\mathbf{K})$ and $\upsilon \in \Sigma_{\rm uc}$ such that $s\upsilon \in \mathcal{L}(\mathbf{L})$. As $s \in \mathcal{L}(\mathbf{K})$ and $\mathbf{K} \subseteq \Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$, it holds that $s \in \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$, which implies $s\upsilon \in \mathcal{L}(\mathbf{K})$. As s and υ were chosen arbitrarily, it follows by Definition 3 that $\mathbf{K}$ is controllable with respect to $\mathbf{L}$. To see that $\mathbf{K} \parallel \mathbf{L}$ is $\Sigma_{\rm p}$-nonblocking, let $s \in \mathcal{L}(\mathbf{K} \parallel \mathbf{L})$. Then $s \in \mathcal{L}(\mathbf{K}) \subseteq \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$, i.e., there exists $t \in \Sigma_{\rm p}^*$ such that $st \in \mathcal{L}^{\omega}(\mathbf{L} \parallel \mathbf{K})$. Thus, $\mathbf{K} \parallel \mathbf{L}$ is $\Sigma_{\rm p}$-nonblocking.

Conversely, assume that $\mathbf{K}$ is controllable with respect to $\mathbf{L}$ and $\mathbf{L} \parallel \mathbf{K}$ is $\Sigma_{\rm p}$-nonblocking, and let $s \in \mathcal{L}(\mathbf{K})$. Let $r \sqsubseteq s$ and $\upsilon \in \Sigma_{\rm uc}$ such that $r\upsilon \in \mathcal{L}(\mathbf{L})$. Then $r \in \mathcal{L}(\mathbf{K})$, and as $\mathbf{K}$ is controllable with respect to $\mathbf{L}$, it follows that $rv \in \mathcal{L}(\mathbf{K})$ and thus $s \in \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$. Further, as $\mathbf{L} \parallel \mathbf{K}$ is $\Sigma_{\rm p}$-nonblocking, for $r \in \mathcal{L}(\mathbf{K}) \subseteq \mathcal{L}(\mathbf{L})$, there exists $t \in \Sigma_{\rm p}^*$ such that $rt \in \mathcal{L}^\omega(\mathbf{L} \parallel \mathbf{K})$, i.e., $s \in \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$. Thus $s \in \theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$, and it follows from (6) that $\mathbf{K} \subseteq \Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$. □

Furthermore, $\Theta_{\mathbf{L},\Sigma_{\rm p}}$ is a monotonic operator on the lattice of Σ-DES.

Proposition 7 Let $\mathbf{L}$, $\mathbf{K}_1$, and $\mathbf{K}_2$ be Σ-DES and $\Sigma_{\rm p} \subseteq \Sigma$. If $\mathbf{K}_1 \subseteq \mathbf{K}_2$ then $\Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_1) \subseteq \Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_2)$.

Proof Assume $\mathbf{K}_1 \subseteq \mathbf{K}_2$. Considering Definition 10, it is enough to show that $\theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_1) \subseteq \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_2)$ and $\theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_1) \subseteq \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_2)$. Firstly, for $s \in \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_1)$, it holds that $s \in \mathcal{L}(\mathbf{K}_1) \subseteq \mathcal{L}(\mathbf{K}_2)$ and for all $r \sqsubseteq s$ and all $v \in \Sigma_{\rm uc}$ such that $rv \in \mathcal{L}(\mathbf{L})$ it holds that $rv \in \mathcal{L}(\mathbf{K}_1) \subseteq \mathcal{L}(\mathbf{K}_2)$, and thus $s \in \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_2)$. Secondly, for $s \in \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_1)$ it holds that $s \in \mathcal{L}(\mathbf{K}_1) \subseteq \mathcal{L}(\mathbf{K}_2)$ and for all $r \sqsubseteq s$ there exists $t \in \Sigma_{\rm p}^*$ such that $rt \in \mathcal{L}^\omega(\mathbf{L} \parallel \mathbf{K}_1) \subseteq \mathcal{L}^\omega(\mathbf{L} \parallel \mathbf{K}_2)$, and thus $s \in \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K}_2)$. □

Proposition 7 shows that $\Theta_{\mathbf{L},\Sigma_{\rm p}}$ is a monotonic operator on the lattice of Σ-DES, so it follows by the Knaster-Tarski theorem [29] that $\Theta_{\mathbf{L},\Sigma_{\rm p}}$ has a greatest fixpoint, which by Proposition 6 is the least restrictive controllable and $\Sigma_{\rm p}$-nonblocking sub-behaviour of $\mathbf{L}$.

To compute the fixpoint in a finite number of steps, it is next shown that the least restrictive controllable and $\Sigma_{\rm p}$-nonblocking sub-behaviour for finite-state deterministic specification $\mathbf{K}$ and plant $\mathbf{L}$ can be computed using the states of the synchronous composition $\mathbf{L} \parallel \mathbf{K}$. Therefore, Definition 12 introduces an iteration on the state set of $\mathbf{L} \parallel \mathbf{K}$, which in Proposition 8 is shown to be equivalent to the above $\Theta_{\mathbf{L},\Sigma_{\rm p}}$.

Definition 11 The *restriction* of $\mathbf{G} = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ to $X \subseteq Q$ is $\mathbf{G}_{|X} = \langle \Sigma, X, \rightarrow_{|X}, Q^\circ \cap X, Q^\omega \cap X \rangle$ where $\rightarrow_{|X} = \{\, (x,\sigma,y) \in \rightarrow \mid x, y \in X \,\}$.

Definition 12 Let $\mathbf{L} = \langle \Sigma, Q_L, \rightarrow_L, Q_L^\circ, Q_L^\omega \rangle$ and $\mathbf{K} = \langle \Sigma, Q_K, \rightarrow_K, Q_K^\circ, Q_K^\omega \rangle$ be two deterministic finite-state automata, and let $\Sigma_{\rm p} \subseteq \Sigma$. The *synthesis step operator* $\bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}} \colon 2^{Q_L \times Q_K} \rightarrow 2^{Q_L \times Q_K}$ for $\mathbf{L}$ and $\mathbf{K}$ with respect to $\Sigma_{\rm p}$ is defined by

$$\bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X) = \bar\theta^{\rm cont}_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X) \cap \bar\theta^{\rm nonb}_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)\,; \tag{8}$$

$$\bar\theta^{\rm cont}_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X) = \{\, (x_L, x_K) \in Q_L \times Q_K \mid \text{for all } v \in \Sigma_{\rm uc} \text{ such that } x_L \xrightarrow{v}_L y_L \text{ there exists } y_K \in Q_K \text{ such that } (x_L, x_K) \xrightarrow{v} (y_L, y_K) \in X \,\}\,;$$

$$\bar\theta^{\rm nonb}_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X) = \{\, (x_L, x_K) \in Q_L \times Q_K \mid (x_L, x_K) \xrightarrow{t}_{|X} (y_L, y_K) \text{ for some } t \in \Sigma_{\rm p}^*,\ y_L \in Q_L^\omega, \text{ and } y_K \in Q_K^\omega \,\}\,.$$

Proposition 8 Let $\mathbf{L} = \langle \Sigma, Q_L, \rightarrow_L, Q_L^\circ, Q_L^\omega \rangle$ and $\mathbf{K} = \langle \Sigma, Q_K, \rightarrow_K, Q_K^\circ, Q_K^\omega \rangle$ be two deterministic finite-state automata, let $\mathbf{S} = \mathbf{L} \parallel \mathbf{K}$, and let $\Sigma_{\rm p} \subseteq \Sigma$. For every state set $X \subseteq Q_L \times Q_K$, it holds that $\Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X}) = \mathbf{S}_{|\bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)}$.

Proof Based on Definition 11 and (2) and (6), it is enough to show $\mathcal{L}(\Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X})) = \mathcal{L}(\mathbf{S}_{|\bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)})$.

First assume that $s \in \mathcal{L}(\Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X})) = \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X}) \cap \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X})$. Then $s \in \mathcal{L}(\mathbf{S}_{|X})$, so there exists a path $\mathbf{S}_{|X} \xrightarrow{s} (x_L, x_K) \in X$. It will be shown that $(x_L, x_K) \in \bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)$. First, for $v \in \Sigma_{\rm uc}$ such that $x_L \xrightarrow{v}_L y_L$, it holds that $sv \in \mathcal{L}(\mathbf{L})$, and since $s \in \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X})$ it follows that $sv \in \mathcal{L}(\mathbf{S}_{|X})$. As $\mathbf{L}$ and $\mathbf{K}$ are deterministic, this implies $(x_L, x_K) \in \bar\theta^{\rm cont}_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)$. Second, as $s \in \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X})$, there exists $t \in \Sigma_{\rm p}^*$ such that $st \in \mathcal{L}^\omega(\mathbf{S}_{|X})$, which by determinism of $\mathbf{L}$ and $\mathbf{K}$ implies $(x_L, x_K) \xrightarrow{t}_{|X} (y_L, y_K) \in Q_L^\omega \times Q_K^\omega$. This shows $(x_L, x_K) \in \bar\theta^{\rm cont}_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X) \cap \bar\theta^{\rm nonb}_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X) = \bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)$. As the same can be shown for all prefixes $r \sqsubseteq s$, it follows that $s \in \mathcal{L}(\mathbf{S}_{|\bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)})$.

Conversely, let $s \in \mathcal{L}(\mathbf{S}_{|\bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)})$, and let $r \sqsubseteq s$. Then $\mathbf{S}_{|\bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)} \xrightarrow{r} (x_L, x_K) \in \bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)$. If $rv \in \mathcal{L}(\mathbf{L})$ for $v \in \Sigma_{\rm uc}$, then as $\mathbf{L}$ is deterministic also $x_L \xrightarrow{v}_L y_L$ for some $y_L \in Q_L$, which given $(x_L, x_K) \in \bar\theta^{\rm cont}_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)$ implies $rv \in \mathcal{L}(\mathbf{S}_{|X})$. Thus $s \in \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X})$. Further, as $(x_L, x_K) \in \bar\theta^{\rm nonb}_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X)$ there exists $t \in \Sigma_{\rm p}^*$ such that $(x_L, x_K) \xrightarrow{t}_{|X} (y_L, y_K) \in Q_L^\omega \times Q_K^\omega$, and thus $s \in \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X})$. Therefore, $s \in \theta^{\rm cont}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X}) \cap \theta^{\rm nonb}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X}) = \mathcal{L}(\Theta_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{S}_{|X}))$. □

By Proposition 8, a language-based step of $\Theta_{\mathbf{L},\Sigma_{\rm p}}$ gives the same result as a state-based step of $\bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}$ when applied to a subset of the states of $\mathbf{L} \parallel \mathbf{K}$. To synthesise the least restrictive controllable and $\Sigma_{\rm p}$-nonblocking sub-behaviour of specification $\mathbf{K}$ with respect to plant $\mathbf{L}$, one first constructs the composition $\mathbf{S} = \mathbf{L} \parallel \mathbf{K}$. Then the iteration

$$X^0 = Q_L \times Q_K \qquad X^{i+1} = \bar\Theta_{\mathbf{L},\mathbf{K},\Sigma_{\rm p}}(X^i) \tag{9}$$

converges against a greatest fixpoint X^n in a finite number of n steps, which by Proposition 8 satisfies $\mathbf{S}_{|X^n} = \sup\mathcal{C}_{\mathbf{L},\Sigma_{\rm p}}(\mathbf{K})$.

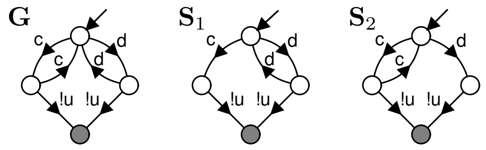

Fig. 5 A DES $\mathbf{G}$ that has no least restrictive supervisor that is nonblocking under control. Events c and d are controllable, while !u is uncontrollable.

5 Related work

This section relates nonblocking with progressive events to other nonblocking conditions studied in the literature.

Multi-tasking supervisory control [5] requires a synthesised supervisor to be nonblocking with respect to several sets of marked states at the same time. *Generalised nonblocking* [6] uses a second set of marked states to specify a subset of the states, from which marked states must be reachable. Both conditions are amenable to synthesis and can be combined with progressive events to further increase modelling capabilities.

The condition of *nonblocking under control* [8] is more similar to that of nonblocking with progressive events. When modelling a supervisor implementation, it is assumed that an implemented supervisor or *controller* sends controllable events as commands to the plant. Typically, the controller can generate several controllable events in quick sequence, and it is considered unlikely that uncontrollable events occur during such a sequence. Then it makes sense to require the system to complete its tasks using Σ_c-*complete* traces.

Definition 13 [8] Let $\mathbf{G} = \langle \Sigma, Q, \to, Q^\circ, Q^\omega \rangle$ and $\Sigma_c \subseteq \Sigma$. The path $x_0 \xrightarrow{\sigma_1} x_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} x_n$ is Σ_c-*complete*, if for each $i = 1, \ldots, n$ it holds that either $\sigma_i \in \Sigma_c$ or there do not exist $\sigma \in \Sigma_c$ and $y \in Q$ such that $x_{i-1} \xrightarrow{\sigma} y$.

Definition 14 [8] Let $\mathbf{G} = \langle \Sigma, Q, \to, Q^\circ, Q^\omega \rangle$ and $\Sigma_c \subseteq \Sigma$. Then $\mathbf{G}$ is *nonblocking under* Σ_c-*control* if for all paths $\mathbf{G} \xrightarrow{s} x$, there exists a Σ_c-complete path $x \xrightarrow{t} y^\omega$ such that $y^\omega \in Q^\omega$.

Nonblocking under control is similar to nonblocking with progressive events, in that it considers uncontrollable events as non-progressive in states where a controllable event is enabled. However, it depends on the state whether an event is progressive or not, and this dependency means that in general there do not exist least restrictive supervisors that are nonblocking under control.

For example, Fig. 5 shows a DES $\mathbf{G}$ which is not nonblocking under control. As the uncontrollable !u-transitions are only enabled in states where controllable events are also enabled, these transitions are considered as non-progressive and cannot be used to prove that the marked state is reachable. The two sub-behaviours $\mathbf{S}_1$ and $\mathbf{S}_2$ are nonblocking under control, however neither of them is least restrictive, and their least upper bound, $\mathbf{G}$, is not nonblocking under control.

It is shown in [26] how the property of nonblocking under control can be verified. Synthesis for this and similar properties can be achieved using ω-languages [9], however these methods do not in general produce a state-based supervisor that can be readily implemented.

6 Conclusions

The condition of nonblocking with progressive events is introduced as an extension of standard nonblocking. It is shown that there are situations where synthesis using the standard nonblocking property results in an unexpected result, because the synthesised supervisor can complete its tasks only if certain rare or undesirable events occur. Using progressive events, it can be specified more precisely how a synthesised supervisor must complete its tasks. The nonblocking property with progressive events of some industrial-scale discrete event systems has been checked using the compositional verification algorithm in Supremica [22], in one case exposing an issue that remains undetected when only the standard nonblocking property is considered. While progressive events increase the modelling capabilities, verification and synthesis can still be achieved without increase in complexity over the standard nonblocking property.

References

[1] Peter J. G. Ramadge and W. Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 1989, 77(1):81–98.

[2] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. New York, NY, USA: Springer Science & Business Media, 2 edition, 2008.

[3] André Arnold. *Finite Transition Systems: Semantics of Communicating Systems*. Hertfordshire, UK: Prentice-Hall, 1994.

[4] Yi-Liang Chen, Stéphane Lafortune, and Feng Lin. Design of nonblocking modular supervisors using event priority functions. *IEEE Transactions on Automatic Control*, 2000, 45(3):432–452.

[5] Max H. de Queiroz, José E. R. Cury, and W. M. Wonham. Multitasking supervisory control of discrete-event systems. *Proceedings of the 7th International Workshop on Discrete Event Systems, WODES '04*. Reims, France: IFAC, 2004: 175–180.

[6] Robi Malik and Ryan Leduc. Generalised nonblocking. *Proceedings of the 9th International Workshop on Discrete Event Systems, WODES'08*. Göteborg, Sweden: IEEE, 2008: 340–345.

[7] Ryan J. Leduc, Bertil A. Brandin, Mark Lawford, and W. M. Wonham. Hierarchical interface-based supervisory control—part I:

Serial case. *IEEE Transactions on Automatic Control*, 2005, 50(9):1322–1335.

[8] P. Dietrich, R. Malik, W. M. Wonham, and B. A. Brandin. Implementation considerations in supervisory control. B. Caillaud, P. Darondeau, L. Lavagno, and X. Xie, editors, *Synthesis and Control of Discrete Event Systems*. Dordrecht, the Netherlands: Kluwer Academic Publishers, 2002: 185–201.

[9] Christine Baier and Thomas Moor. A hierarchical control architecture for sequential behaviours. *Proceedings of the 11th International Workshop on Discrete Event Systems, WODES'12*. Guadalajara, Mexico: IFAC, 2012: 259–264.

[10] Simon Ware and Robi Malik. Supervisory control with progressive events. *Proceedings of the 11th IEEE International Conference on Control and Automation, ICCA 2014*. Taichung, Taiwan: IEEE, 2014: 1461–1466.

[11] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Boston, MA, USA: Addison-Wesley, 2001.

[12] W. M. Wonham. Supervisory control of discrete-event systems. Systems Control Group, Department of Electrical Engineering, University of Toronto, Ontario, Canada, http://www.control.utoronto.edu/, 2009.

[13] R. Su and W. Murray Wonham. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 2004, 14(1):31–53.

[14] Bertil A. Brandin, Robi Malik, and Petra Malik. Incremental verification and synthesis of discrete-event systems guided by counter-examples. *IEEE Transactions on Control Systems Technology*, 2004, 12(3):387–401.

[15] Susanne Graf and Bernhard Steffen. Compositional minimization of finite state systems. *Proceedings of the 1990 Workshop on Computer-Aided Verification*, volume 531 of *LNCS*. New Brunswick, NJ, USA: Springer, 1990: 186–196.

[16] Antti Valmari. Compositionality in state space verification methods. *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, volume 1091 of *LNCS*. Osaka, Japan: Springer, 1996: 29–56.

[17] Hugo Flordal and Robi Malik. Compositional verification in supervisory control. *SIAM Journal of Control and Optimization*, 2009, 48(3):1914–1938.

[18] P. N. Pena, J. E. R. Cury, and S. Lafortune. Verification of nonconflict of supervisors using abstractions. *IEEE Transactions on Automatic Control*, 2009, 54(12):2803–2815.

[19] Rong Su, Jan H. van Schuppen, Jacobus E. Rooda, and Albert T. Hofkamp. Nonconflict check by using sequential automaton abstractions based on weak observation equivalence. *Automatica*, 2010, 46(6):968–978.

[20] Simon Ware and Robi Malik. Conflict-preserving abstraction of discrete event systems using annotated automata. *Discrete Event Dynamic Systems: Theory and Applications*, 2012, 22(4):451–477.

[21] Robi Malik, David Streader, and Steve Reeves. Conflicts and fair testing. *International Journal of Foundations of Computer Science*, 2006, 17(4):797–813.

[22] Knut Åkesson, Martin Fabian, Hugo Flordal, and Robi Malik. Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*. Ann Arbor, MI, USA: IEEE, 2006: 384–385.

[23] Robi Malik and Ryan Leduc. Compositional nonblocking verification using generalised nonblocking abstractions. *IEEE Transactions on Automatic Control*, 2013, 58(8):1–13.

[24] Bertil Brandin and François Charbonnier. The supervisory control of the automated manufacturing system of the AIP. *Proceedings of Rensselaer's 4th International Conference on Computer Integrated Manufacturing and Automation Technology*. Troy, NY, USA: IEEE Computer Society Press, 1994: 319–324.

[25] Ryan James Leduc. *Hierarchical Interface-based Supervisory Control*. PhD thesis, Department of Electrical Engineering, University of Toronto, Ontario, Canada, 2002.

[26] Petra Malik. *From Supervisory Control to Nonblocking Controllers for Discrete Event Systems*. PhD thesis, University of Kaiserslautern, Kaiserslautern, Germany, 2003.

[27] Annika Hinze, Petra Malik, and Robi Malik. Interaction design for a mobile context-aware system using discrete event modelling. *Proceedings of the 29th Australasian Computer Science Conference, ACSC '06*. Hobart, Australia: Australian Computer Society, 2006: 257–266.

[28] KORSYS Project, http://www4.in.tum.de/proj/korsys/.

[29] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 1955, 5(2):285–309.

Simon WARE received his Bachelor of Computing and Mathematical Sciences degree with Honours from the University of Waikato in Hamilton, New Zealand in 2007. Also in 2007, he was involved in a project for discrete event simulation of port biosecurity procedures at AgResearch in Hamilton. He received his Ph.D. in computer science from the University of Waikato in 2014. He is currently a research fellow at Nanyang Technological University in Singapore. His main research interests are liveness and fairness properties of 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.