Generalised Nonblocking

Robi Malik
Department of Computer Science
University of Waikato, Hamilton, New Zealand
robi@cs.waikato.ac.nz

Ryan Leduc
Department of Computing and Software
McMaster University, Hamilton, Canada
leduc@mcmaster.ca

***Abstract*— This paper studies the nonblocking check used in supervisory control of discrete event systems and its limitations. Different examples with different liveness requirements are discussed. It is shown that the standard nonblocking check can be used to specify most requirements of interest, but that it lacks expressive power in a few cases. A generalised nonblocking check is proposed to overcome the weakness, and its relationship to standard nonblocking is explored. Results suggest that generalised nonblocking, while having the same useful properties with respect to synthesis and compositional verification, can provide for more concise problem representations in some cases.**

I. Introduction

Blocking or *conflicts* are common faults in the design of concurrent programs that can be very subtle and hard to detect [1], [2]. They have long been studied in the field of *discrete event systems* [3], [4], which is applied to the modelling of complex, safety-critical systems [5]–[7]. To improve the reliability of such systems, techniques are needed to facilitate the design of nonblocking systems.

A discrete event system is nonblocking if, from every reachable state, all involved components *can* cooperatively reach a terminal state. It is not required that a terminal state is necessarily reached on every possible execution path, only that there exists an execution path to a terminal state. This weak liveness condition has been used very successfully for the synthesis of *discrete event systems* [3], [4], [8]. Other works investigate the compositional semantics [9], [10] of nonblocking and its compositional verification [11], [12].

Despite its widespread use, the expressive powers of nonblocking are limited. To overcome some of the weaknesses, nonblocking has been generalised to handle multiple marking conditions [13], [14]. During their attempts to develop compositional verification methods for the conditions of *hierarchical interface-based supervisory control* [15], the authors discovered a nonblocking-like verification problem that still is difficult to cast into standard nonblocking.

In an attempt to pave the way for compositional verification [11] of a wider range of properties, this paper proposes a more general nonblocking condition, which includes the original nonblocking condition as a special case. Sect. II introduces the notation for finite-state automata and a formal definition of the nonblocking property. Sect. III presents four examples of liveness verifications problems that occur in discrete event systems modelling and discusses how nonblocking can be used to address them. In Sect. IV, the generalised nonblocking property is introduced and shown to cover all the examples, and results about synthesis and compositional verification are given. Finally, Sect. V contains some concluding remarks.

II. Preliminaries

In this paper, discrete event systems are modelled using nondeterministic automata. While most of the concepts concerning nonblocking can be explained using deterministic automata, nondeterminism is needed for compositional verification in Sect. IV-E.

Event sequences and languages are a simple means to describe system behaviours. Their basic building blocks are *events*, which are taken from a finite *alphabet* Σ. In addition, the *silent event* $\tau \notin \Sigma$ is used, with the notation $\Sigma_\tau = \Sigma \cup \{\tau\}$.

Σ^* denotes the set of all finite *strings* of the form $\sigma_1\sigma_2 \dots \sigma_n$ of events from Σ, including the *empty string* ε. The *concatenation* of two strings $s, t \in \Sigma^*$ is written as st. A subset $\mathcal{L} \subseteq \Sigma^*$ is called a *language*. For $\Omega \subseteq \Sigma$, *natural projection* P_Ω: $\Sigma^* \to \Omega^*$ denotes the operation that deletes all events not in Ω from strings.

Definition 1: A (nondeterministic) *automaton* is a tuple $G = \langle \Sigma, X, \rightarrow, X^\circ, X^m \rangle$ where Σ is a finite set of *events*, X is a set of *states*, $\rightarrow \subseteq X \times \Sigma_\tau \times X$ is the *state transition relation*, $X^\circ \subseteq X$ is the set of *initial states*, and $X^m \subseteq X$ is the set of *marked states*.

Definition 2: An automaton $G = \langle \Sigma, X, \rightarrow, X^\circ, X^m \rangle$ is *deterministic* if X° is a singleton, $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$, and $\rightarrow$ contains no τ-transitions.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to strings in Σ_τ^* in the standard way. For state sets $X_1, X_2 \subseteq X$, the notation $X_1 \xrightarrow{s} X_2$ denotes the existence of $x_1 \in X_1$ and $x_2 \in X_2$ such that $x_1 \xrightarrow{s} x_2$. Also, $x \rightarrow y$ denotes the existence of a string $s \in \Sigma_\tau^*$ such that $x \xrightarrow{s} y$, and $x \xrightarrow{s}$ denotes the existence of a state $y \in X$ such that $x \xrightarrow{s} y$. Finally, $G \rightarrow x$ stands for $X^\circ \rightarrow x$.

Synchronous composition and *hiding* are common operations to manipulate languages and automata. Synchronous composition models parallel execution and is done using lock-step synchronisation in the style of [16]. Hiding is an elementary operation of abstraction.

Definition 3: Let $G_1 = \langle \Sigma_1, X_1, \rightarrow_1, X_1^\circ, X_1^m \rangle$ and $G_2 = \langle \Sigma_2, X_2, \rightarrow_2, X_2^\circ, X_2^m \rangle$ be two automata. The *synchronous product* $G_1 \parallel G_2$ of G_1 and G_2 is

$$\langle \Sigma_1 \cup \Sigma_2, X_1 \times X_2, \rightarrow, X_1^\circ \times X_2^\circ, X_1^m \times X_2^m \rangle \qquad (1)$$

where

$(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2)$ if $\sigma \in (\Sigma_1 \cap \Sigma_2), x_1 \xrightarrow{\sigma}_1 y_1, x_2 \xrightarrow{\sigma}_2 y_2$;
$(x_1, x_2) \xrightarrow{\sigma} (y_1, x_2)$ if $\sigma \in (\Sigma_1 \cup \{\tau\}) \setminus \Sigma_2, x_1 \xrightarrow{\sigma}_1 y_1$;
$(x_1, x_2) \xrightarrow{\sigma} (x_1, y_2)$ if $\sigma \in (\Sigma_2 \cup \{\tau\}) \setminus \Sigma_1, x_2 \xrightarrow{\sigma}_2 y_2$.

Definition 4: Let $G = \langle \Sigma, X, \rightarrow, X^\circ, X^m \rangle$ be an automaton, and let $\Upsilon \subseteq \Sigma$. The result of *hiding* Υ in G is

$$G \setminus \Upsilon \ = \ \langle \Sigma \setminus \Upsilon, X, \rightarrow \setminus \Upsilon, X^\circ, X^m \rangle \ , \qquad (2)$$

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

It is a desirable property that every execution of an automaton can be completed by reaching a marked state in X^m, otherwise *livelock* or *deadlock* may occur. In discrete event systems theory, an automaton that is unable to terminate in this way is called *blocking*. This concept becomes more interesting when several automata are running in parallel—in this case the term *conflicting* is also used [4]. The following extends the standard definition [4] to the case of nondeterministic automata considered in this paper.

Definition 5: An automaton $G = \langle \Sigma, X, \rightarrow, X^\circ, X^m \rangle$ is called *nonblocking*, if for all states $x \in X$ such that $G \rightarrow x$, it also holds that $x \rightarrow X^m$.

In order to be nonblocking, it is sufficient that a terminal state *can* be reached in every possible situation. This is equivalent to termination under an implicit *strong fairness* assumption stating that "whenever a transition can occur infinitely often, it occurs infinitely often" [17].

III. Applications

This section discusses four examples where nonblocking checks are used to verify liveness properties of interest.

A. Dining Philosophers

The dining philosophers problem [18] is an illustrative example of a common computing problem in concurrency. Five philosophers are sitting at a circular table with a large bowl of spaghetti in the centre. A fork is placed between each pair of philosophers, and it is assumed that each philosopher must eat with the two forks next to him. Each philosopher can be either *thinking* or *eating*, and the objective is to ensure that every philosopher can eventually get a chance to eat.

The simplest approach to verify the absence of starvation in this system is to consider the state where all philosophers are *thinking* as the success state. Yet, this is a much too weak property, since a system in which all philosophers keep thinking indefinitely would be nonblocking.

The desired liveness property in this system is that each philosopher can always enter the *eating* state. However, the dynamics of the systems does not permit all philosophers to be eating at the same time. Yet, it is possible to consider the set of states, where philosopher 1 is eating (independently of the states of the other philosophers) as success states, and perform a nonblocking check with respect to this marking condition. By repeating the same check for each of the five philosophers, the absence of starvation in the entire system can be verified.

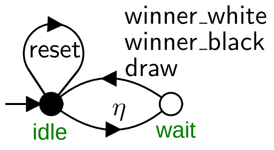

Fig. 1. Verifying the simple game nonblocking.

In this example, a single nonblocking check is not enough to verify the liveness requirements of a system. The possibility of handling multiple termination conditions simultaneously and its use in synthesis is discussed in [14].

B. Simple Game

Assume that a simple board game is to be modelled. Two players are taking their moves in turn, modifying the game state with each move. It is a desired liveness property of this system that the game can always end, i.e., it should always be possible that one player wins, or that a draw is declared.

This can easily be achieved by marking all the states where one player has won, or the game is over without a winner. Then a standard nonblocking check can be carried out to verify the desired property that the game can always end.

To complicate the example slightly, a reset feature is added to the model: an additional event reset is introduced which can always be executed and resets the game to its initial state.

With this addition, the standard nonblocking check becomes much less expressive. Indeed, there may be states where one player refuses to perform any further move, so the game cannot end. However, due the omnipresent possibility of reset, the system is still nonblocking as long as there is only some way of ending the game from its initial state.

In this modified model, it is much more interesting to verify whether *"the game can always end, even if* reset *is not used."* This stronger property can be verified using a standard nonblocking check, if the model is modified by adding the automaton in Fig. 1. Here, η is a new event that does not occur anywhere in the model, and which therefore can occur at any time. This event is used only for the nonblocking check. It can be understood as an observer, who at any time can temporarily disable reset in order to check whether the system can terminate successfully without it.

C. Nonblocking under Control

A similar verification problem is discussed in [1]. When modelling reactive systems, there typically are two types of events, *controllable* events that are in some way produced by the control software, and *uncontrollable* events that spontaneously occur in the system to be controlled.

In this context, a nonblocking check is used to verify whether the controlled system can always terminate. However, it may be desirable to rule out certain unlikely behaviours and require that termination can always be achieved by "expected" or "normal" behaviour. In [1], it is considered unlikely that uncontrollable events occur while the controller is in the process of sending a sequence of commands. More formally, termination is required to be achievable by means of a *complete* sequence of events.

Definition 6: Let $G = \langle \Sigma, X, \to, X^\circ, X^m \rangle$ be a deterministic automaton, and let $\Sigma_c \subseteq \Sigma$. The path

$$x_0 \xrightarrow{\sigma_1} x_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} x_n \tag{3}$$

is called Σ_c*-complete* in G, if for each $i = 1, \ldots, n$ it holds that either $\sigma_i \in \Sigma_c$ or there does not exist any $\sigma \in \Sigma_c$ such that $x_{i-1} \xrightarrow{\sigma}$.

Definition 7: Let $G = \langle \Sigma, X, \to, X^\circ, X^m \rangle$ be a deterministic automaton, and let $\Sigma_c \subseteq \Sigma$. G is said to be *nonblocking under* Σ_c*-control* if for all states $x \in X$ such that $G \to x$, there exists a Σ_c-complete path $x \to X^m$ in G.

It is discussed in [7] how this property can be expressed and verified using a standard nonblocking check. Essentially, a similar translation as in the previous example is used, but with more sophisticated modelling to restrict the possible behaviours to be Σ_c-complete once the η event has occurred.

D. Hierarchical Interface-Based Supervisory Control

Yet another type of nonblocking property occurs in hierarchical interface-based supervisory control [15]. In this context, systems are decomposed into *high-level* and *low-level* modules in a master-slave relationship. An interface between the modules is defined, and properties are verified by considering only one module and the interface at a time. One of these properties is given in the following definition. For a full discussion of the remaining properties and the context, the reader is referred to Def. 5 in [15].

Definition 8: Let $\Sigma = \Sigma_R \dot\cup \Sigma_A \dot\cup \Sigma_L$, and let $I = \langle \Sigma_R \cup \Sigma_A, X_I, \to_I, \{x_I^\circ\}, X_I^m \rangle$ and $L = \langle \Sigma, X_L, \to_L, \{x_L^\circ\}, X_L^m \rangle$ be two deterministic automata. I and L are said to satisfy *Serial Interface Consistency (SIC) Property V* if, for all strings $s \in \Sigma^*$ and all events $\rho \in \Sigma_R$ such that $s\rho \in \mathcal{L}(I \| L)$, and for all events $\sigma \in \Sigma_A$ such that $P_{\Sigma_R \cup \Sigma_A}(s)\rho\sigma \in \mathcal{L}(I)$, there exists $t \in \Sigma_L^*$ such that $s\rho t\sigma \in \mathcal{L}(I \| L)$.

The definition says that, after the occurrence of *request* ρ, all *answers* σ that are possible according to the interface I must somehow be possible to occur in the system. However, once the request has occurred, the low level L may make decisions and evolve in ways that preclude the occurrence of some of the answers that were originally possible.

SIC property V is similar to the standard nonblocking property if states where answer event σ is possible are considered as marked states. However, instead of requiring a path from all reachable states to these states, such a path is required only from states *immediately* after request event ρ. Thus, although the property clearly is related to nonblocking, it is weaker than standard nonblocking, and cannot be expressed easily as a standard nonblocking problem. The following section proposes the generalised nonblocking condition that can be used to cover this property also.

IV. Generalised Nonblocking

A. Formal Definition

To model more general nonblocking properties, it is convenient to specify different state sets of automata and define relationships between them. Therefore, automata are extended to *multi-coloured* automata by labelling states with different *colours* or *propositions*. Afterwards, the generalised nonblocking condition is defined using these propositions.

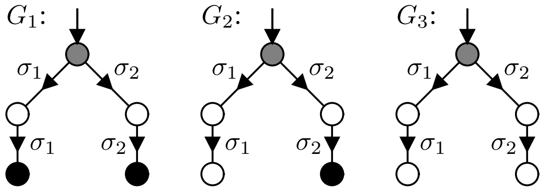

Fig. 2. Generalised nonblocking vs. standard nonblocking.

Definition 9: A *multi-coloured automaton* is a tuple $G = \langle \Sigma, \Pi, X, \to, X^\circ, \Xi \rangle$ where Σ is a finite set of *events*, Π is a finite set of *propositions* or *colours*, $\to \subseteq X \times \Sigma_\tau \times X$ is the *state transition relation*, $X^\circ \subseteq X$ is the set of *initial states*, and $\Xi \colon \Pi \to 2^X$ defines the set of marked states for each proposition in Π.

This definition allows different sets of marked states for each proposition. The concept is similar to that of Kripke-structures [19], but the above definition is inspired by [14]. The main difference to [14] is the colouring map Ξ, which maps propositions to state sets and not vice versa, to allow for slightly more concise notation in this paper.

An automaton without colours, $G = \langle \Sigma, X, \to, X^\circ, X^m \rangle$, can be considered as a multi-coloured automaton with a single proposition ω by defining $\Xi(\omega) = X^m$. Most standard notations and definitions for automata such as state transitions, languages, and hiding are naturally lifted to multi-coloured automata. In synchronous composition, marking for all colours needs to be considered.

Definition 10: Let $G_1 = \langle \Sigma_1, \Pi, X_1, \to_1, X_1^\circ, \Xi_1 \rangle$ and $G_2 = \langle \Sigma_2, \Pi, X_2, \to_2, X_2^\circ, \Xi_2 \rangle$ be two multi-coloured automata. The synchronous product of G_1 and G_2 is

$$G_1 \| G_2 = \langle \Sigma, \Pi, X \to, X^\circ, \Xi \rangle \tag{4}$$

where Σ, X, $\to$, and X° are given as in Def. 3, and $\Xi(\pi) = \Xi_1(\pi) \times \Xi_2(\pi)$ for each $\pi \in \Pi$.

The generalised nonblocking property uses two propositions, called α and ω. The intended meaning is that ω represents terminal states and corresponds to the traditional marked states, while α specifies a set of states from which marked states are required to be reachable.

Definition 11: Let $G = \langle \Sigma, \Pi, X, \to, X^\circ, \Xi \rangle$ with $\alpha, \omega \in \Pi$ be a multi-coloured automaton. G is called (α, ω)*-nonblocking*, if for all states $x \in X$ such that $G \to x$ and $x \in \Xi(\alpha)$ it also holds that $x \to \Xi(\omega)$.

Generalised nonblocking requires that, from all reachable states marked α, it is possible to reach a state marked ω. Clearly, if an automaton is standard nonblocking (Def. 5), it is also (α, ω)-nonblocking. The following example shows the the converse is not true in general.

Example 1: Consider the automata in Fig. 2. States marked α are grey, and states marked ω are black. Only automaton G_3 is (α, ω)-blocking, although G_2 and G_3 are blocking according to Def. 5 if all states marked ω are considered as terminal states.

B. Relationship to Standard Nonblocking

Generalised nonblocking includes standard nonblocking as a special case. Clearly, if all states of an automaton are marked by the proposition α, then (α, ω)-nonblocking requires that a state marked ω is reached from every state, i.e., that the automaton is nonblocking in the sense of Def. 5. This translation from standard nonblocking to generalised nonblocking is straightforward and does not change the size or complexity of the automaton translated.

The reverse translation from generalised nonblocking to standard nonblocking is much more involved. For such a translation to be of interest, it should be applicable to a set of composed automata without first constructing their synchronous product, and without first checking the (α, ω)-nonblocking property. The following translation can be done in a modular way and highlights the major problems that need to be overcome.

Definition 12: Let $G = \langle \Sigma, \Pi, X, \rightarrow, X^\circ, \Xi \rangle$ be a multi-coloured automaton with $\alpha, \omega \in \Pi$. The *conflict-automaton* $G_{\text{conf}} = \langle \Sigma_{\text{conf}}, X_{\text{conf}}, \rightarrow_{\text{conf}}, X^\circ_{\text{conf}}, X^m_{\text{conf}} \rangle$ for G is constructed as follows.

- $\Sigma_{\text{conf}} = \Sigma \cup \{\alpha, \beta, \omega\}$, where $\alpha, \beta, \omega \notin \Sigma$;
- $X_{\text{conf}} = X \times (\Xi(\alpha) \cup \{\perp\})$, where $\perp \notin X$;
- $\rightarrow_{\text{conf}}$ consists of the following transitions:

$$\begin{aligned} (x, r) &\xrightarrow{\sigma}_{\text{conf}} (y, r), && \text{if } x \xrightarrow{\sigma} y\ ; \\ (x, \perp) &\xrightarrow{\alpha}_{\text{conf}} (x, x), && \text{if } x \in \Xi(\alpha)\ ; \\ (x, r) &\xrightarrow{\beta}_{\text{conf}} (r, r), && \text{if } r \neq \perp\ ; \\ (x, r) &\xrightarrow{\omega}_{\text{conf}} (x, \perp), && \text{if } x \in \Xi(\omega) \text{ and } r \neq \perp\ ; \end{aligned}$$

- $X^\circ_{\text{conf}} = \{\, (x, \perp) \mid x \in X^\circ \,\}$.
- $X^m_{\text{conf}} = \{\, (x, \perp) \mid x \in X \,\}$.

The idea of this translation is as follows. Propositions α and ω are introduced as events, enabled in the states marked with these propositions. The initial states are marked; transitions to an unmarked state are only possible via an α event, and after the occurrence of α, a marked state can only be reached when ω occurs. However, to ensure that only the states *immediately* after α are checked for nonblocking, the *reset event* β is added to all states reached later and provides a transition back to the state immediately after α. In this way, it is ensured that automaton G_2 in Fig. 2 is translated to a nonblocking automaton, and only G_3 is translated to a blocking automaton.

Proposition 1: Let $G = \langle \Sigma, \Pi, X, \rightarrow, X^\circ, \Xi \rangle$ be a multi-coloured automaton with $\alpha, \omega \in \Pi$. Then G is (α, ω)-nonblocking if and only if G_{conf} is nonblocking.

Proof: First let G be (α, ω)-nonblocking, and let $G_{\text{conf}} \xrightarrow{s}_{\text{conf}} (x, r)$. If $r = \perp$, then $(x, r) \in X^m_{\text{conf}}$ and therefore $(x, r) \xrightarrow{\varepsilon}_{\text{conf}} X^m_{\text{conf}}$. Otherwise, if $r \neq \perp$, the string s must contain the event α by construction and can be written as $s = s'\alpha t'$ for some $t' \in (\Sigma \cup \{\beta\})^*$. Again by construction,

$$G_{\text{conf}} \xrightarrow{s'\alpha}_{\text{conf}} (r, r) \xrightarrow{t'}_{\text{conf}} (x, r) \tag{5}$$

where $r \in \Xi(\alpha)$. By removing all maximal substrings $u\beta$, $u \in \Sigma^*$, and all α and ω events from s', a string $s'' \in \Sigma^*$ can be constructed such that $G \xrightarrow{s''} r \in \Xi(\alpha)$. Since G is (α, ω)-nonblocking, $r \xrightarrow{t} y \in \Xi(\omega)$ for some $t \in \Sigma^*$. Thus,

$$(x, r) \xrightarrow{\beta}_{\text{conf}} (r, r) \xrightarrow{t}_{\text{conf}} (y, r) \xrightarrow{\omega}_{\text{conf}} (y, \perp) \in X^m_{\text{conf}} \tag{6}$$

by construction, i.e., G_{conf} is nonblocking.

Second let G_{conf} be nonblocking, and let $G \xrightarrow{s} x \in \Xi(\alpha)$. It follows by construction that

$$G_{\text{conf}} \xrightarrow{s}_{\text{conf}} (x, \perp) \xrightarrow{\alpha}_{\text{conf}} (x, x)\ . \tag{7}$$

Since G_{conf} is nonblocking, there exists $t \in (\Sigma \cup \{\alpha, \beta, \omega\})^*$ such that $(x, x) \xrightarrow{t}_{\text{conf}} X^m_{\text{conf}}$. By construction, t must contain the event ω and can be written as $t = t'\omega u'$ for some $t' \in (\Sigma \cup \{\beta\})^*$. This means that

$$(x, x) \xrightarrow{t'}_{\text{conf}} (y, x) \xrightarrow{\omega}_{\text{conf}} (y, \perp) \tag{8}$$

for some $y \in \Xi(\omega)$. If $t' \in \Sigma^*$, it follows immediately that $x \xrightarrow{t'} y \in \Xi(\omega)$. Otherwise, t' contains the event β and can be written as $t' = u''\beta t''$ for some $t'' \in \Sigma^*$. Then

$$(x, x) \xrightarrow{u''\beta}_{\text{conf}} (x, x) \xrightarrow{t''}_{\text{conf}} (y, x) \xrightarrow{\omega}_{\text{conf}} (y, \perp)\ , \tag{9}$$

which implies $x \xrightarrow{t''} y$. In both cases, $y \in \Xi(\omega)$ can be reached from x, i.e., G is (α, ω)-nonblocking. ■

The translation from generalised nonblocking to standard nonblocking given above increases the number of states by a factor proportional to the number of states marked α in the original automaton. It is unknown to the authors whether there exists a more efficient translation with the same properties, although evidence so far suggests that this is not possible. This is in line with similar results about automata on infinite words [20], [21].

On the other hand, generalised nonblocking can be verified at least as easily as standard nonblocking: it suffices to check for each reachable state marked α (as opposed to each reachable state for standard nonblocking) that a state marked ω can be reached. This suggests that verifying generalised nonblocking directly is preferable to translation to standard nonblocking with the associated blow-up.

C. Expressing SIC Property V

SIC Property V can be expressed directly using generalised nonblocking. This is achieved by marking precisely those states α that are entered immediately after a request event.

More precisely, let automata I and L and event $\sigma \in \Sigma_A$ be given as in Def. 8. Then multi-coloured automata I^σ, L^σ, and T^σ are constructed such that $I^\sigma \parallel L^\sigma \parallel T^\sigma$ is (α, ω)-nonblocking if and only if SIC Property V is satisfied for the given answer σ. The construction is as follows.

- I^σ is obtained from I by adding propositions α and ω such that all states with σ enabled are marked α, and all states are marked ω. If $I = \langle \Sigma_R \cup \Sigma_A, X_I, \rightarrow_I, X^\circ_I, X^m_I \rangle$, then $I^\sigma = \langle \Sigma_R \cup \Sigma_A, \Pi, X_I, \rightarrow_I, X^\circ_I, \Xi_I \rangle$ with $\Pi = \{\alpha, \omega\}$, $\Xi_I(\alpha) = \{\, x \in X_I \mid x \xrightarrow{\sigma} \,\}$, and $\Xi_I(\omega) = X_I$.

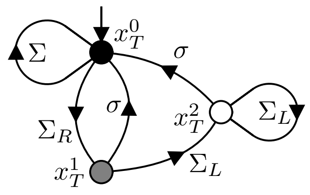

Fig. 3. The automaton T^σ for translating SIC Property V into (α,ω)-nonblocking.

- L^σ is obtained from L by marking all states with both α and ω. If $L = \langle \Sigma, X_L, \to_L, X_L^\circ, X_L^m \rangle$, then $L^\sigma = \langle \Sigma, \Pi, X_L, \to_L, X_L^\circ, \Xi_L \rangle$ with $\Xi_L(\alpha) = \Xi_L(\omega) = X_L$.
- $T^\sigma = \langle \Sigma, \Pi, X_T, \{x_T^0\}, \to_T, \Xi_T \rangle$ with $X_T = \{x_T^0, x_T^1, x_T^2\}$, $\Xi_T(\alpha) = \{x_T^1\}$ and $\Xi_T(\omega) = \{x_T^0\}$ is the nondeterministic multi-coloured automaton in Fig. 3.

Proposition 2: Let $\Sigma = \Sigma_R \mathbin{\dot\cup} \Sigma_A \mathbin{\dot\cup} \Sigma_L$, and let automata I and L be given as in Def. 8. For each $\sigma \in \Sigma_A$ construct multi-coloured automata I^σ, L^σ, and T^σ as explained above. I and L satisfy SIC Property V if and only if $I^\sigma \parallel L^\sigma \parallel T^\sigma$ is (α,ω)-nonblocking for each $\sigma \in \Sigma_A$.

Proof: First, let I and L satisfy SIC Property V. Let $\sigma \in \Sigma_A$ such that $I^\sigma \parallel L^\sigma \parallel T^\sigma \xrightarrow{s} (x_I, x_L, x_T) \in \Xi_I(\alpha) \times \Xi_L(\alpha) \times \Xi_T(\alpha)$. Since $\Xi_T(\alpha) = \{x_T^1\}$ by construction, it holds that $x_T = x_T^1$ and $s = s'\rho$ for some $s' \in \Sigma^*$ and $\rho \in \Sigma_R$. Also $I \| L \xrightarrow{s} (x_I, x_L)$, and since $x_I \in \Xi_I(\alpha)$, it holds by construction that $I \xrightarrow{P_{\Sigma_R \cup \Sigma_A}(s)} x_I \xrightarrow{\sigma} y_I$ for some $y_I \in X_I = \Xi_I(\omega)$. Therefore, $s'\rho = s \in \mathcal{L}(I \parallel L)$ and $P_{\Sigma_R \cup \Sigma_A}(s')\rho\sigma \in \mathcal{L}(I)$. By SIC Property V, there exists $t \in \Sigma_L^*$ such that $st\sigma = s'\rho t\sigma \in \mathcal{L}(I \parallel L)$. Since L is deterministic, there exists $y_L \in X_L = \Xi_L(\omega)$ such that $x_L \xrightarrow{t\sigma} y_L$. Furthermore, note that $x_T = x_T^1 \xrightarrow{t\sigma} x_T^0 \in \Xi_T(\omega)$ for any string $t \in \Sigma_L^*$. Therefore, $(x_I, x_L, x_T) \xrightarrow{t\sigma} \Xi_I(\omega) \times \Xi_L(\omega) \times \Xi_T(\omega)$, i.e., $I^\sigma \parallel L^\sigma \parallel T^\sigma$ is (α,ω)-nonblocking.

Second, let $I^\sigma \| L^\sigma \| T^\sigma$ be (α,ω)-nonblocking for all $\sigma \in \Sigma_A$. Let $s \in \Sigma^*$, $\rho \in \Sigma_R$, and $\sigma \in \Sigma_A$ be such that $s\rho \in \mathcal{L}(I \| L)$ and $P_{\Sigma_R \cup \Sigma_A}(s)\rho\sigma \in \mathcal{L}(I)$. Then there exist states $x_I \in X_I$ and $x_L \in X_L$ such that $I \xrightarrow{P_{\Sigma_R \cup \Sigma_A}(s)\rho} x_I \xrightarrow{\sigma}$ and $L \xrightarrow{s\rho} x_L$. Also, $T^\sigma \xrightarrow{s\rho} x_T^1$ by construction of T^σ, and therefore $I^\sigma \parallel L^\sigma \parallel T^\sigma \xrightarrow{s\rho} (x_I, x_L, x_T^1) \in \Xi_I(\alpha) \times \Xi_L(\alpha) \times \Xi_T(\alpha)$. Since $I^\sigma \parallel L^\sigma \parallel T^\sigma$ is (α,ω)-nonblocking, there exists $u \in \Sigma^*$ such that $(x_I, x_L, x_T^1) \xrightarrow{u} \Xi_I(\omega) \times \Xi_L(\omega) \times \Xi_T(\omega)$. By construction of T^σ, and since $\Xi_T(\omega) = \{x_T^0\}$, there exists a prefix $t\sigma$ of u such that $t \in \Sigma_L^*$ and $(x_I, x_L, x_T^1) \xrightarrow{t\sigma}$, i.e., $s\rho t\sigma \in \mathcal{L}(I \parallel L)$. Since s, ρ, and σ have been chosen arbitrarily, I and L satisfy SIC Property V. ■

The above construction can be performed modularly, i.e., if I and L are composed of several automata, then the construction can be applied to the individual components. Unlike Def. 12, the state space is only increased by a constant factor of three. This is a small price for a construction that makes SIC Property V amenable to compositional verification as in [11]—to the best of the authors' knowledge, this property so far has only been verified using explicit or symbolic state-space exploration [22].

D. *Synthesis*

Having defined the concept of (α,ω)-nonblocking, it is of interest to study its properties. A crucial question in supervisory control theory is whether synthesis is feasible. Given a language $\mathcal{L}$, the problem of *synthesis* consists of finding a sublanguage $\mathcal{L}' \subseteq \mathcal{L}$ that satisfies particular properties of interest such as controllability or nonblocking [4].

To discuss synthesis in the context of nondeterministic automata, the concept of subautomata and union of automata are used. The following definitions are adapted from [23].

Definition 13: Let $G = \langle \Sigma, \Pi, X, \to, X^\circ, \Xi \rangle$ and $G' = \langle \Sigma, \Pi, X, \to_{G'}, X_{G'}^\circ, \Xi \rangle$ be two automata with the same alphabets and state sets. G' is a *subautomaton* of G, written $G' \subseteq G$, if $\to_{G'} \subseteq \to$ and $X_{G'}^\circ \subseteq X^\circ$.

Definition 14: Let $G_j = \langle \Sigma, \Pi, X, \to_j, X_j^\circ, \Xi \rangle$, $j \in J$ be a family of automata all having the same alphabets and state sets. The *union* of the automata G_j is defined as

$$\bigcup_{j \in J} G_j \;=\; \langle \Sigma, \Pi, X, \bigcup_{j \in J} \to_j, \bigcup_{j \in J} X_j^\circ, \Xi \rangle \;. \qquad (10)$$

To simplify notation, all automata are assumed to have the same state sets and markings. This is exactly the situation encountered in synthesis. Unreachable states can always be removed, but this is not discussed here.

Given an automaton G, there typically are several subautomata G' that qualify as a solution to a particular synthesis problem, and the question arises which one to choose. Here it is desirable to identify a *most general* or *least restrictive* solution, which can be used as a unique synthesis result. The obvious candidate for this is the union of all solutions, provided that the result of combining two or more solutions to the synthesis problem still is a solution. This requirement is known to be satisfied for controllability and for nonblocking of deterministic automata [4]. It is easy to show that it also holds for generalised nonblocking of nondeterministic automata.

Proposition 3: Let $G_j = \langle \Sigma, \Pi, X, \to_j, X_j^\circ, \Xi \rangle$, $j \in J$ be a family of automata with $\alpha, \omega \in \Pi$ such that each G_j is (α,ω)-nonblocking. Then $\bigcup_{j \in J} G_j$ is (α,ω)-nonblocking.

Every nondeterministic automaton has a maximal (α,ω)-nonblocking subautomaton. This result can be combined with results about controllability or other properties [4], [23] and used to build synthesis algorithms.

E. *Compositional Verification*

The straightforward approach to verify whether a composed system

$$G_1 \parallel G_2 \parallel \cdots \parallel G_n \qquad (11)$$

is (α,ω)-nonblocking consists of explicitly constructing the synchronous composition and checking whether a state marked ω can be reached from every state marked α. This can be done using CTL model checking, and models of substantial size can be analysed if the state space is represented symbolically [19]. Yet, the technique remains limited by the amount of memory available to store representations of the synchronous product.
]

As an alternative, compositional reasoning [10] attempts to rewrite individual components of a composed system such as (11) and, e.g., replace G_1 by a simpler version G'_1, and then analyse the simpler system

$$G'_1 \parallel G_2 \parallel \cdots \parallel G_n \ . \tag{12}$$

Such compositional reasoning requires that G_1 and G'_1 are related in some way. An appropriate notion of equivalence has been identified for nonblocking verification in [10], and these results can easily be adapted to the case of (α, ω)-nonblocking considered in this paper.

Definition 15: Let G_1 and G_2 be two multi-coloured automata with $\alpha, \omega \in \Pi$. Then G_1 and G_2 are called (α, ω)-*nonblocking equivalent*, written $G_1 \simeq_{(\alpha,\omega)} G_2$, if for any multi-coloured automaton T with the same proposition set Π, it holds that $G_1 \| T$ is (α, ω)-nonblocking if and only if $G_2 \| T$ is (α, ω)-nonblocking.

To be feasible for compositional verification as discussed above, the equivalence used must be well-behaved with respect to synchronous composition and hiding. These so-called *congruence* properties can easily be shown for (α, ω)-nonblocking equivalence.

Proposition 4: Let G_1, G_2, T be multi-coloured automata with $\alpha, \omega \in \Pi$. If $G_1 \simeq_{(\alpha,\omega)} G_2$, then $G_1 \| T \simeq_{(\alpha,\omega)} G_2 \| T$.

Proposition 5: Let $G = \langle \Sigma, \Pi, X, \rightarrow, X^\circ, \Xi \rangle$ be a multi-coloured automaton with $\alpha, \omega \in \Pi$, and let $\Upsilon \subseteq \Sigma$. Then G is (α, ω)-nonblocking if and only if $G \setminus \Upsilon$ is (α, ω)-nonblocking.

Note that, if given two automata G and H such that H does not use any events in alphabet Υ, then $(G \parallel H) \setminus \Upsilon = (G \setminus \Upsilon) \parallel H$. In combination with Prop. 5 this means that abstractions can be applied in a compositional way, as long as only events local to the subsystem considered are abstracted away. Subsystems can be simplified individually or composed as needed, and the verification and simplification strategies outlined in [10], [11] can be used.

Observation equivalence, which comes with efficient simplification algorithms [24], can be shown to preserve (α, ω)-nonblocking equivalence. Furthermore, the fact that generalised nonblocking is a weaker property than standard nonblocking can make more aggressive simplification possible than for standard nonblocking.

V. Conclusions

A generalised nonblocking condition has been introduced as a simple extension of standard nonblocking that makes it possible to express certain other nonblocking-like properties concisely. Generalised nonblocking permits synthesis and compositional verification like standard nonblocking does, and can be verified with the same computational effort. The framework for compositional verification outlined in this paper is the first known method to make SIC property V [15] amenable to modular verification. This framework can now be used to develop a unified approach for standard nonblocking, as well as other properties such as SIC property V and controllability [23].

References

[1] P. Dietrich, R. Malik, W. M. Wonham, and B. A. Brandin, "Implementation considerations in supervisory control," in *Synthesis and Control of Discrete Event Systems*, B. Caillaud, P. Darondeau, L. Lavagno, and X. Xie, Eds. Kluwer, 2002, pp. 185–201.

[2] K. C. Wong, J. G. Thistle, R. P. Malhame, and H.-H. Hoang, "Supervisory control of distributed systems: Conflict resolution," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 10, pp. 131–186, 2000.

[3] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer, Sept. 1999.

[4] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[5] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Trans. Automat. Contr.*, vol. 38, no. 7, pp. 1040–1059, July 1993.

[6] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," in *Proc. Rensselaer's 4th Int. Conf. Computer Integrated Manufacturing and Automation Technology*, Troy, NY, USA, 1994, pp. 319–324.

[7] P. Malik, "From supervisory control to nonblocking controllers for discrete event systems," Ph.D. dissertation, University of Kaiserslautern, Kaiserslautern, Germany, 2003.

[8] Y.-L. Chen, S. Lafortune, and F. Lin, "Design of nonblocking modular supervisors using event priority functions," *IEEE Trans. Automat. Contr.*, vol. 45, no. 3, pp. 432–452, Mar. 2000.

[9] R. Kumar and M. A. Shayman, "Non-blocking supervisory control of nondeterministic discrete event systems," in *Proc. American Control Conf.*, Baltimore, MD, USA, 1994, pp. 1089–1093.

[10] R. Malik, D. Streader, and S. Reeves, "Conflicts and fair testing," *Int. J. Found. Comput. Sci.*, vol. 17, no. 4, pp. 797–813, 2006.

[11] H. Flordal and R. Malik, "Modular nonblocking verification using conflict equivalence," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, MI, USA, July 2006, pp. 100–106.

[12] P. N. Pena, J. E. R. Cury, and S. Lafortune, "New results on testing modularity of local supervisors using abstractions," in *Proc. 11th IEEE Int. Conf. Emerging Technologies and Factory Automation, ETFA '06*, Prague, Czech Republic, Sept. 2006, pp. 950–956.

[13] M. Fabian and R. Kumar, "Mutually nonblocking supervisory control of discrete event systems," in *Proc. 36th IEEE Conf. Decision and Control, CDC '97*, San Diego, CA, USA, 1997, pp. 2970–2975.

[14] M. H. de Queiroz, J. E. R. Cury, and W. M. Wonham, "Multi-tasking supervisory control of discrete-event systems," in *Proc. 7th Int. Workshop on Discrete Event Systems, WODES '04*, Reims, France, Sept. 2004, pp. 175–180.

[15] R. J. Leduc, B. A. Brandin, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control—part I: Serial case," *IEEE Trans. Automat. Contr.*, vol. 50, no. 9, pp. 1322–1335, Sept. 2005.

[16] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.

[17] A. Arnold, *Finite Transitions Systems: Semantics of Communicating Systems*. Prentice-Hall, 1994.

[18] E. W. Dijkstra, "Hierarchical ordering of sequential processes," *Acta Inf.*, vol. 1, no. 2, pp. 115–138, 1971.

[19] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.

[20] Q. Yan, "Lower bounds for complementation of ω-automata via the full automata technique," *Logical Methods in Computer Science*, vol. 4, no. 1:5, pp. 1–20, 2008.

[21] S. Safra, "On the complexity of ω-automata," Ph.D. dissertation, Weizmann Inst. of Science, Rehovot, Israel, 1989.

[22] R. Song and R. J. Leduc, "Symbolic synthesis and verification of hierarchical interface-based supervisory control," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, MI, USA, July 2006, pp. 419–426.

[23] H. Flordal and R. Malik, "Supervision equivalence," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, MI, USA, July 2006, pp. 155–160.

[24] R. Paige and R. E. Tarjan, "Three partition refinement algorithms," *SIAM J. Computing*, vol. 16, no. 6, pp. 973–989, 1987.