Compositional Verification of Non-Blockingness with Prioritised Events

Yiheng Tang * **Thomas Moor** *

* *Institute of automatic control, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany (e-mail: lrt@fau.de)*

Abstract: This paper addresses the verification of non-blockingness for modular discrete-event systems, i.e., discrete-event systems that are composed from component models. For such systems, the explicit construction of a monolithic representation turns out intractable for relevant applications, since such a construction in general is of exponential cost w.r.t. the number of components. One well established approach to circumvent the need for a monolithic representation for the verification task at hand is to alternate (a) the substitution of individual components by abstractions and (b) the composition of only a small number of strategically chosen components at a time. When successful, one ends up with a single moderately sized automaton which does not represent the overall behaviour in any detail but which does block if and only if the original modular system fails to be non-conflicting. This approach is referred to as *compositional verification* and originates from the field of process algebra with more recent adaptations to finite automata models. The main contribution of the present study is the development of a number of abstraction rules valid for compositional verification of non-conflictingness in the presence of global event priorities, i.e., where high priority events from one component possibly preempt events with lower priority of other components.

Keywords: discrete-event systems, compositional verification, non-conflicting, non-blocking, event priorities, modular systems

1. INTRODUCTION

Considering discrete-event systems that are representable as finite automata, a well studied liveness property is *non-blockingness*, i.e., the ability of the system to attain an accepted configuration from any reachable state. For example, in the context of supervisory control theory (Ramadge and Wonham (1987)), where marked states are used to represent task-completion, non-blockingness is a desired closed-loop property.

For a moderately sized single automaton, non-blockingness can be verified by a straightforward enumeration-based reachability analysis. This approach to verification can also be applied to modular systems represented as the synchronous composition (Milner (1989)) of a number of component models since the overall behaviour can be represented again as a single automaton. The construction of such a monolithic representation, however, does not scale well with the number of components and for relevant applications more often than not turns out infeasible when using enumeration-based procedures. This contrasts the fact that the implementation of the overall behaviour by computer software and hardware, e.g. by a *programmable logic controller (PLC)* in an industrial automation context, does not suffer from this issue since this does not require a monolithic representation; see e.g. LRT/FGDES (2019). This motivates the interest in methods for the verification of non-blockingness for modular discrete-event systems that likewise circumvent an explicit monolithic representation.

One well established approach to address this situation is referred to as *compositional verification*. Inspired by test theory (Nicola and Hennessy (1984)), compositional non-blockingness verification attempts to abstract each component model while preserving non-blockingness when synchronising with any arbitrary other *test-automaton*. This of course includes the special case of the test-automaton to be the synchronous composition of the remaining component models. Such an abstraction is called *conflict equivalent*. Specifically for automata representations of component models, various qualifying abstraction rules have been proposed in the literature; see e.g. Flordal and Malik (2009); Su et al. (2010); Ware and Malik (2012); Pilbrow and Malik (2015). Once abstraction rules have been applied to the component models, one strategically chooses a small number of components and substitutes them with their actual synchronous composition. While this increases the state count, it also potentially decreases the number of shared events. In turn, a subsequent substitution via conflict equivalent abstractions is expected to again decrease the state count. The two forms of substitutions are then alternated until only one automaton is left. The latter automaton is tested for non-blockingness by e.g. enumeration-based methods and, by using only conflict preserving abstractions, the result carries over to the original modular system. Clearly, one does not expect to beat computational complexity and the overall procedure may need to be aborted due to exceeding available computational resources. However, the above cited literature demonstrates by a number of practical case studies

the applicability to relevant large scale systems with an impressive computational performance.

In the present paper, we consider the situation where, besides the synchronous composition of all automata, the global behaviour of the system is additionally affected by event priorities (Lüttgen (1998); Cleaveland et al. (2007)). In this scenario, each event is associated with an integer attribute to represent its priority. At any global state, each event should be disabled if an event with higher priority is currently enabled. In particular, this includes the case where the preempting event is private to some other module. Our main technical contribution are a number of abstraction rules which turn out conflict equivalent and, hence, can be used for compositional verification with event priorities.

We envisage two main use cases for our findings. First, consider the verification of a control algorithm that is implemented by a *programmable logic controller (PLC)*. A common approach here is to preprocess the PLC code to obtain a more formal representation. Inspecting the semantics of *sequential function charts (SFCs)* as specified in IEC 61131-3, we observe distinguished classes of events with different priorities to preempt each other, e.g. reading from line levels, execution of activity code, reconfiguration of tokens, writing to line levels; see also Bauer et al. (2004) for formal SFC semantics. Similar considerations apply to *activity diagrams (ADs)* as defined by the *unified modelling language (UML)* and *sequential behaviour diagrams (SBDs)* defined by the *Interdisciplinary Modelling Language (IML)*; see Eshuis (2007) and Brecher et al. (2016) for ADs and SBDs, respectively. For a second use case, consider the scenario where a modular discrete-event system has been synthesised by formal methods to enforce a language inclusion specification next to a non-blocking closed loop. Here, we may after the fact want to introduce priorities to achieve a consistent and repeatable behaviour. This restricts event execution and, hence, we need to verify whether our assignment of priorities maintains the non-blockingness guaranteed by the original design. This is of particular interest when implementing a modular supervisor by converting the component models to executable PLC code using a code generator which explicitly or implicitly assigns priorities; see e.g. LRT/FGDES (2019); Fabian and Hellgren (1998); Qamsane et al. (2016).

The paper is organised as follows. In Section 2, we clarify preliminaries and notation. A formal definition of prioritised events including the semantics thereof and a suitable adaptation of conflict equivalence for compositional verification is given in Section 3. A number of conflict equivalent abstractions w.r.t. prioritised events are developed in Sections 4 and 5 to constitute our main technical contribution. In Section 6, we evaluate our results in the context of two examples. Observing applicable page constraints, we omit technical proofs in this paper and make them available as a technical report instead; see Tang and Moor (2022)

2. PRELIMINARIES

We recall some common notation regarding finite automata as relevant for the present paper.

An *alphabet* A is a finite set of symbols, also referred to as *events*. Given an alphabet A, its *Kleene closure* A^* denotes the set of all finite strings, i.e., sequences of events. By convention, A^* includes the empty string $\epsilon \notin A$. The *concatenation* of two strings $s, t \in A^*$ is written $st \in A^*$.

A *non-deterministic finite automaton over* A is a tuple $G := \langle Q, A, \rightarrow, Q^\circ \rangle$ with the finite *state set* Q, the *transition relation* $\rightarrow \subseteq Q \times A \times Q$, and the set of *initial states* $Q^0 \subseteq Q$. Using infix form, we write $x \xrightarrow{\alpha} y$ or $x \not\xrightarrow{\alpha} y$ whenever $(x, \alpha, y) \in \rightarrow$ or $(x, s, y) \notin \rightarrow$, respectively. Throughout this paper, when using the infix form for a relation, left out parameters are interpreted as existential quantification, e.g., the expression $(x \xrightarrow{\alpha})$ evaluates true if and only if $(\exists y : x \xrightarrow{\alpha} y)$. For a state x, the set of *enabled events* is denoted $G(x) := \{\alpha \in A \mid x \xrightarrow{\alpha}\}$. A sequence of states related via transitions is referred to as a *trace*, written $x_0 \xrightarrow{\alpha_1} x_1 \xrightarrow{\alpha_2} x_2 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_k} x_k$, or, when the intermediate states are regarded not relevant, more concisely $x_0 \xrightarrow{\alpha_1}\xrightarrow{\alpha_2}\xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_k} x_k$ or $x \xrightarrow{s} y$ with $x = x_0$, $s = \alpha_1\alpha_2\cdots\alpha_k$ and $y = x_k$. This effectively extends the transition relation to string-valued labels in the common way. Here, we stipulate $x \xrightarrow{\epsilon} x$ for all $x \in Q$. For further notational convenience, we write $X \xrightarrow{s} Y$ with $X, Y \subseteq Q$ if there exist $x \in X$ and $y \in Y$ such that $x \xrightarrow{s} y$. Likewise, $X \xrightarrow{s}$ and $G \xrightarrow{s}$ are short forms for $X \xrightarrow{s} Q$ and $Q^0 \xrightarrow{s} Q$, respectively. We say $x \in Q$ is *reachable* if $G \xrightarrow{s} x$.

Regarding termination, we consider the distinguished *termination event* $\omega \in A$ and require the existence of a unique terminal state $x^{\mathrm{T}} \in Q$ with the properties (i) for all $x \xrightarrow{\alpha} y$ we have that $y = x^{\mathrm{T}}$ if and only if $\alpha = \omega$, and (ii) $x^{\mathrm{T}} \xrightarrow{\omega} x^{\mathrm{T}}$. In graphical representations, predecessors of x^{T} are depicted as full black circles and ω-transitions are omitted. A state $x \in Q$ is *co-reachable* if it can be continued to attain x^{T}, i.e., if there exists $s \in A^*$ such that $x \xrightarrow{s} x^{\mathrm{T}}$. The latter condition is equivalent to $x \xrightarrow{s\omega}$. An automaton G is *non-blocking* if all reachable states are co-reachable. Provided that the state count of G is not too high, non-blockingness can be verified by enumeration based methods, i.e., by explicitly computing the sets of reachable and co-reachable states, respectively.

3. PRIORITISED EVENTS AND COMPOSITIONAL VERIFICATION

When the behaviour of an automaton shall be implemented by a physical device, and if in some state multiple transitions are enabled for execution, the implementation somehow needs to decide which transition to take. Associating a priority with each event is one option here, and extending the execution semantics of automata in this regard renders this implementation detail explicit, e.g, in order to analyse its effects with formal methods. Vice versa, if we set up a discrete-event system to model the behaviour of some physical implementation, the latter may be constructed to execute transitions based on event priorities. Here, a semantical extension allows us to more directly obtain the respective automata models. In this section, we begin by providing a formal definition of automata execution semantics with event priorities, and proceed by inspecting some basic consequences for modu-

lar systems, i.e., when the overall behaviour is represented by the composition of multiple automata.

We consider a *universe of events* U together with a (not necessarily injective) map $\mathsf{prio}\colon U \to \mathbb{N}$ to assign a priority to each individual event. From now on, we will implicitly assume $A \subseteq U$ for any alphabet relevant for our study. Note that we read priorities as ordinal numbers, i.e., $1 \in \mathbb{N}$ for *first priority*, $2 \in \mathbb{N}$ for *second priority*, and so on: the lower the number, the higher the priority, with the highest priority 1. Specifically, "$\mathsf{prio}(\sigma) < \mathsf{prio}(\rho)$" reads "$\sigma$ is of higher priority than ρ". For notational convenience, we introduce the following short forms regarding priorities for an alphabet $A \subseteq U$ and an automaton G, respectively:

(a) events of priority higher than $n \in \mathbb{N}$
$A^{<n} := \{\alpha \in A \mid \mathsf{prio}(\alpha) < n\}$;
(b) events of priority higher than $\mathsf{prio}(\alpha)$ for $\alpha \in U$
$A^{<\alpha} := A^{<\mathsf{prio}(\alpha)}$;
(c) lowest priority within A
$\mathsf{lo}(A) := \mathsf{max}\{\mathsf{prio}(\alpha) \mid \alpha \in A\}$;
(d) enabled events at state x with priority above $n \in \mathbb{N}$
$G^{<n}(x) := G(x) \cap U^{<n}$.

We shall now formally represent the behavioural restriction caused by event priorities imposed on an automaton $G = \langle Q, A, \to, Q^\circ \rangle$. In any state x, if some event α is enabled, it preempts any transition labeled by an event α' with lower priority, i.e., with $\mathsf{prio}(\alpha) < \mathsf{prio}(\alpha')$. The following shaping operator removes the affected transitions.

Definition 1. Given an automaton $G = \langle Q, A, \to, Q^\circ \rangle$, the *shaping operator* $\mathcal{S}(\cdot)$ is defined $\mathcal{S}(G) := \langle Q, A, \to^{\mathcal{S}}, Q^\circ \rangle$ where $x \xrightarrow{\alpha}{}^{\mathcal{S}} y$ if and only if $x \xrightarrow{\alpha} y$ and $G^{<\alpha}(x) = \emptyset$. □

With this definition, $\mathcal{S}(G)$ represents the behaviour of G with prioritised events as specified by the map $\mathsf{prio}\colon U \to \mathbb{N}$. It should be noted that shaping can turn a blocking automaton into a non-blocking one and vice versa. Hence, to verify non-blockingness of a system with event priorities, we may first set up G, second apply $\mathcal{S}(\cdot)$ and finally perform a reachability analysis, e.g. on enumeration basis.

We now turn to a variation of the common synchronous composition in order to address modular systems with event priorities. Technically, we refer to a disjoint union composition $U = \Sigma \dot\cup \Upsilon$ of our universe of events, with Σ the *regular events* and Υ the *silent events*. The latter are not subject to synchronisation. Since termination is meant to be synchronous, we have $\omega \in \Sigma$. Moreover, we assume that for each regular event $\sigma \in \Sigma$ there exists a unique silent event $\tau \in \Upsilon$ with matching priority and this event is denoted $\tau =: \mathsf{hide}(\sigma)$; i.e., we have $\mathsf{hide}\colon \Sigma \to \Upsilon$ with $\mathsf{prio}(\mathsf{hide}(\sigma)) = \mathsf{prio}(\sigma)$. In graphical representations, we use the convention $\tau_{(n)} := \mathsf{hide}(\sigma)$ for $\sigma \in \Sigma$ with $\mathsf{prio}(\sigma) = n$. The decomposition $U = \Sigma \dot\cup \Upsilon$ and the semantics to be introduced in the sequel are seen as a generalisation of a *single distinguished silent event* $\Upsilon = \{\tau\}$, as commonly used in the context of compositional verification without event priorities; see e.g. Flordal and Malik (2009).

Before proceeding, we introduce additional notational conventions for a concise reference:

(a) the natural projection denoted $\mathsf{p}\colon U^* \to \Sigma^*$ removes silent events from strings in U^*, see e.g., Cassandras and Lafortune (2008) for a formal definition;
(b) the abstract transition relation $\Rightarrow \subseteq Q \times \Sigma^* \times Q$, defined by $x \overset{s}{\Rightarrow} y$ for $s \in \Sigma^*$ if and only if there exists some $s' \in U^*$ such that $\mathsf{p}(s') = s$ and $x \xrightarrow{s'} y$;
(c) we may omit explicit intermediate states, e.g., we write $x \xrightarrow{s}\overset{t}{\Rightarrow} y$ as a short form for the existence of $z \in Q$ such that $x \xrightarrow{s} z$ and $z \overset{t}{\Rightarrow} y$;
(d) a trace is *silent* if all its event labels belong to Υ;
(e) enabled silent events (with priority above $n \in \mathbb{N}$)
$G_{\mathrm{slnt}}(x) := G(x) \cap \Upsilon$; $G^{<n}_{\mathrm{slnt}}(x) := G^{<n}(x) \cap \Upsilon$;
(f) enabled regular events (with priority above $n \in \mathbb{N}$)
$G_{\mathrm{rglr}}(x) := G(x) - \Upsilon$; $G^{<n}_{\mathrm{rglr}}(x) := G^{<n}(x) - \Upsilon$.

Considering the composition of two specific automata over alphabets A_1 and A_2, respectively, $A_1 \cap A_2 \cap \Sigma$ are called the *shared events*, while all other events from $A_1 \cup A_2$ are *private events*. By the following definition, the composition of two automata will synchronise the execution of shared events while allowing private events to be executed independently.

Definition 2. Given two automata $G_1 = \langle Q_1, A_1, \to_1, Q_1^0 \rangle$ and $G_2 = \langle Q_2, A_2, \to_2, Q_2^0 \rangle$, their synchronous composition is defined by

$$G_1 \parallel G_2 := \langle Q_1 \times Q_2, A_1 \cup A_2, \to, Q_1^0 \times Q_2^0 \rangle$$

where $(x_1, x_2) \xrightarrow{\alpha} (y_1, y_2)$ if and only if one of the following three conditions is satisfied:

$$\begin{aligned}
&\alpha \in (A_1 \cap A_2) - \Upsilon, \quad x_1 \xrightarrow{\alpha}_1 y_1, \quad x_2 \xrightarrow{\alpha}_2 y_2\,; \\
&\alpha \in (A_1 - A_2) \cup \Upsilon, \quad x_1 \xrightarrow{\alpha}_1 y_1, \quad x_2 = y_2\,; \\
&\alpha \in (A_2 - A_1) \cup \Upsilon, \quad x_1 = y_1, \quad x_2 \xrightarrow{\alpha}_2 y_2\,. \qquad \square
\end{aligned}$$

As with the plain synchronous composition, the above variation is commutative and distributive up to bijective renaming of states; i.e., $G_1 \parallel G_2$ equals $G_2 \parallel G_1$ after suitable reordering of state tuples; likewise $(G_1 \parallel G_2) \parallel G_3$ equals $G_1 \parallel (G_2 \parallel G_3)$ after suitable reordering. In particular, we may drop parenthesis in the latter two expressions. It should be noted that the synchronous composition of non-blocking automata may turn out blocking and vice versa. If a modular system is given in the form $M = G_1 \parallel G_2 \parallel \cdots \parallel G_n$, we may first explicitly evaluate the synchronous composition and subsequently perform a reachability analysis to test for non-blockingness. Since the state count of M is exponential in the number n of components, this direct approach becomes intractable even for a moderate number of components. Here the so called *compositional verification* comes into play. This approach suggests to abstract individual modules by automata with potentially smaller state count and/or smaller transition count but without affecting the blockingness of the overall composition. Such abstractions are called *conflict equivalent*. The abstraction stage is then alternated with the composition of a small number of strategically chosen modules until only one module is left. The latter can be inspected by e.g. an enumeration based reachability analysis. This approach was demonstrated to achieve impressive computational performance for various practical large scale case studies; see Flordal and Malik (2009) and Pilbrow and Malik (2015).

Now consider again a modular system $M = G_1 \parallel G_2 \parallel \cdots \parallel G_n$, however, with event priorities as defined above.

Here, we would like to verify non-blockingness of $\mathcal{S}(M)$. In other words, we consider event priority as having a global effect on M, e.g., a high priority event in one component is meant to preempt lower-priority events in other components.

Definition 3. A family of automata $(G_i)_{1\leq i\leq n}$ is *non-conflicting w.r.t. prioritised events* if $\mathcal{S}(G_1 \parallel G_2 \parallel \cdots \parallel G_n)$ is non-blocking. □

For the scope of the present paper, the above property is also concisely referred to as *non-conflicting* and it is precisely this property, that we seek to verify in an efficient manner. If it was that the shaping operator distributed over the synchronous composition, we could utilize exactly the same abstraction methods as those established for the situation without event priorities. Unfortunately this is not the case and, for our situation, a suitable notion of conflict equivalence will need to explicitly refer to $\mathcal{S}(\cdot)$. Since the synchronous composition is commutative, we focus attention without loss of generality on an abstraction of G_1. Technically, we consider the situation of

$$\mathcal{S}(\underbrace{G_1}_{:=G} \parallel \underbrace{G_2 \parallel \cdots \parallel G_k}_{:=H}), \tag{1}$$

and ask for an abstraction G' of G such that $\mathcal{S}(G' \parallel H)$ is non-blocking if and only if $\mathcal{S}(G \parallel H)$ is so.

A first and rather simplistic candidate for a suitable abstraction is to obtain G' from G by relabelling any transition with a private but regular event $\sigma \in \Sigma$ by its silent counterpart $\mathsf{hide}(\sigma)$. This substitution is referred to as *hiding of private events*. It is immediate from Definitions 1 and 2 that this abstraction does not affect blockingness in the shaped product with one and the same automaton H. Thus, from now on we will assume without loss of generality that all private events of G in $G||H$ are silent.

A second and likewise simple candidate for a suitable abstraction is to obtain G' from G by shaping w.r.t. silent events only. Given $G = \langle Q, A, \rightarrow, Q^\circ\rangle$, we define the Υ*-shaping operator* by $\mathcal{S}_\Upsilon(G) := \langle Q, A, \rightarrow^{\mathcal{S}_\Upsilon}, Q^\circ\rangle$ where $x \xrightarrow{\alpha}{}^{\mathcal{S}_\Upsilon} y$ if $x \xrightarrow{\alpha} y$ and $G^{<\alpha}_{\mathrm{slnt}}(x) = \emptyset$. In other words, $\mathcal{S}_\Upsilon(\cdot$ discards all transitions which are preempted by a silent transition. As an immediate consequence from Definitions 1 and 2 we obtain $\mathcal{S}(G' \parallel H) = \mathcal{S}(G \parallel H)$ for the abstraction $G' = \mathcal{S}_\Upsilon(G)$. In particular, this abstraction does not affect blockingness in the shaped product with any automaton H. Thus, from now on we will assume without loss of generality that G is Υ-shaped, i.e., that $G = \mathcal{S}_\Upsilon(G)$.

Most relevant for practical purposes, the two abstraction rules identified so far can be applied without the potentially intractable evaluation of the transition relation of $H = G_2 \parallel \cdots \parallel G_k$. This concept is made explicit in the following formal definition of *conflict equivalence w.r.t. event priorities*, which, as in e.g. Flordal and Malik (2009); Mohajerani et al. (2014), is inspired by test-theory; see e.g. Nicola and Hennessy (1984).

Definition 4. Two automata G and G' are *conflict equivalent w.r.t. prioritised events*, denoted $G \simeq_{\mathcal{S}} G'$, if for any automaton T, G and T are non-conflicting w.r.t. prioritised events if and only if G' and T are non-conflicting w.r.t. prioritised events. □

For the scope of the present paper, the above property is also concisely referred to as *conflict equivalence*. Although not explicitly required, G and the abstraction G' are expected to refer to the same alphabet – otherwise one could too easily construct a test-automaton to invalidate conflict equivalence. We summarise our assumptions for easy reference.

Assumption 5. Whenever discussing the product $G \parallel T$, we assume implicitly and without loss of generality that all private events are silent and that G is Υ-shaped by suitable pre-processing. For the sake of a concise notation, we indicate states from G with a subscript $(\cdot)_G$ and states from T with a subscript $(\cdot)_T$ and assume all state sets to be disjoint. In consequence, we can at most instances omit the respective subscripts for transition relations, since e.g. $x_G \xrightarrow{\alpha} y_G$ implies the transition to be in G. □

4. ABSTRACTION RULES BASED ON PRIORITISED WEAK BISIMULATION

A generic approach to obtain an abstraction with reduced state count of an automaton $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ is to consider an equivalence relation $\sim \subseteq Q \times Q$ on Q and to merge states per equivalence class $[x] := \{x' \in Q \,|\, (x, x') \in \sim\}$ to obtain the so called *quotient automaton*. For our situation with prioritised silent events, an adaptation that addresses silent live-locks turns out useful.

Definition 6. Given a Υ-shaped automaton $G = \langle Q, A, \rightarrow, Q^\circ\rangle$, an *n-live-lock* in G is a silent trace

$$x_0 \xrightarrow{\tau_1} x_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_k} x_k = x_0 \tag{2}$$

where $k \geq 1$, $\mathsf{lo}(\{\tau_1, \cdots, \tau_k\}) = n$ and for any $i \in \{1, \cdots, k\}$, $x \in Q$ and $\tau \in \Upsilon$, $x_i \xrightarrow{\tau} x$ implies that there exists some $j \in \{1, \cdots, k\}$ so that $x = x_j$. □

We use the shorthand τ-live-lock to denote $\mathsf{prio}(\tau)$-live-lock for $\tau \in \Upsilon$. Technically, a live-lock is a silent strongly connected component where none of the states can leave this strongly connected component by executing a silent transition. Due to event priority, live-locks may indefinitely *trap* other automata under synchronisation, i.e. when in an n-live-lock of some automaton G, a synchronised automaton T can never execute a silent event τ with $\mathsf{prio}(\tau) > n$. With this notion of n-live-locks, we define the *quotient automata* as follows.

Definition 7. Given a Υ-shaped automaton $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ and an equivalence relation $\sim \subseteq Q \times Q$, the *quotient automaton* $G/\!\sim$ of G w.r.t. $\sim$ is defined by $G/\!\sim := \langle Q/\!\sim, A, \rightarrow_\sim, \tilde{Q}^\circ, M\rangle$ where $Q/\!\sim = \{[x] \,|\, x \in Q\}$, $\tilde{Q}^\circ = \{[x^\circ] \,|\, x^\circ \in Q^\circ\}$ and

$$\rightarrow_\sim = \{[x] \xrightarrow{\alpha} [y] \,|\, x \xrightarrow{\alpha} y\} - \{[x] \xrightarrow{\tau} [x] \,|\, \tau \in \Upsilon \text{ and not all states of a } \tau\text{-live-lock in } G \text{ are in } [x]\}$$ □

Comparing with the conventional quotient automata construction, our definition avoids introducing inexistent live-locks while the existent live-locks are still preserved. This potentially renders the trapping power before and after abstraction consistent. It is also worth mentioning that if a state appears in two live-locks in a Υ-shaped automaton, then both live-locks must visit the same set of states.

Example 8. Let $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ be such as given in Figure 1 with $\Sigma = \{\sigma, \omega\}$ and Q be partitioned by an

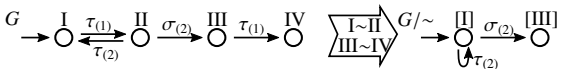

Fig. 1. quotient with post-processed silent transitions

equivalence relation $\sim \subseteq Q \times Q$ into the two classes $\{\mathrm{I}, \mathrm{II}\}$ and $\{\mathrm{III}, \mathrm{IV}\}$. The corresponding $G/\!\sim$ is given on the right side of Figure 1. Note that I and II form a 2-live-lock while III and IV are not in any live-lock in G. □

Based on the conventional process algebra CCS (Milner (1989)), Lüttgen (1998) introduced the variant CCS$^{\mathrm{ch}}$ to model concurrent systems with global event priority. In fact, the semantics inferred by a shaped automaton in our framework are quite similar to the operational semantics of CCS$^{\mathrm{ch}}$. By extending the well-known *weak bisimulation* from CCS, Lüttgen (1998) defines the *prioritised weak bisimulation* (PWB) as a reasoning framework in CCS$^{\mathrm{ch}}$. Following the convention in Lüttgen (1998), we first distinguish certain types of transitions.

Definition 9. Given a Υ-shaped automaton $G = \langle Q, A, \rightarrow, Q^\circ \rangle$, we define two more transition relations for $\Delta \subseteq \Sigma$:

(T1) $\xrightarrow[\Delta:n]{} \subseteq Q \times A \times Q$ with $x \xrightarrow[\Delta:n]{\alpha} y$ if and only if $x \xrightarrow{\alpha} y$ and $G^{<n}_{\mathrm{rglr}}(x) \subseteq \Delta$;

(T2) $\Longrightarrow_{\Delta:n} \subseteq Q \times \{\epsilon\} \times Q$ with $x \Longrightarrow_{\Delta:n} y$ if and only if $x \xrightarrow[\Delta:n]{\tau_1} \cdots \xrightarrow[\Delta:n]{\tau_k} y$, $k \geq 0$ and $\tau_1 \cdots \tau_k \in (\Upsilon^{<(n+1)})^*$. □

Comparing with $\Rightarrow$, the restricted transition relation $\Longrightarrow_{\Delta:n}$ accounts for silent transitions that cannot be preempted by events in Δ with priority higher than n. Note that specifically for any state x, any priority value $n \in \mathbb{N}$ and any event set $\Delta \subseteq \Sigma$, we always have $x \xrightarrow[\Delta:n]{\epsilon} x$ and $x \Longrightarrow_{\Delta:n} x$, which slightly extends (T1). This enables the definition of PWB for the context of the present paper.

Definition 10. Let $G = \langle Q, A, \rightarrow, Q^\circ \rangle$ be a Υ-shaped automaton. A symmetric relation $\approx \subseteq Q \times Q$ is a *prioritised weak bisimulation on* G (PWB) if for any $x, x' \in Q$ so that $x \approx x'$, the following hold:

(P1) If $G^{<n}_{\mathrm{slnt}}(x) = \emptyset$ for some $n \geq 0$, then there exists y' so that $x \approx y'$, $G^{<n}_{\mathrm{slnt}}(y') = \emptyset$, $G^{<n}_{\mathrm{rglr}}(y') \subseteq \Delta$ and $x' \Longrightarrow_{\Delta:n} y'$ where $\Delta = G^{<n}_{\mathrm{rglr}}(x)$;

(P2) If $x \xrightarrow{\alpha} y$, then there exists y' so that $y \approx y'$ and $x' \Longrightarrow_{\Delta:\alpha} \xrightarrow[\Delta:\alpha]{\mathsf{p}(\alpha)} \Longrightarrow_{\Sigma:1} y'$ where $\Delta = G^{<\alpha}_{\mathrm{rglr}}(x)$. □

In the original literature Lüttgen (1998), PWB as a binary relation over automata has been shown to be a congruence w.r.t. composition "|" and restriction "$/L$". Thus, by a similar line of thought as in Malik et al. (2004), two PWB automata turn out conflict equivalent. In the context of the present paper, we obtain the following result.

Theorem 11. Let $G = \langle Q, A, \rightarrow, Q^\circ \rangle$ be a Υ-shaped automata with a PWB $\approx \subseteq Q \times Q$. It then holds that $G \simeq_{\mathcal{S}} (G/\!\approx)$. □

Note that PWB is defined such that if at some state a regular event $\sigma \in \Sigma$ can be executed, an equivalent state must be also able to execute σ, either immediately or after a number of silent steps with priority *not lower*

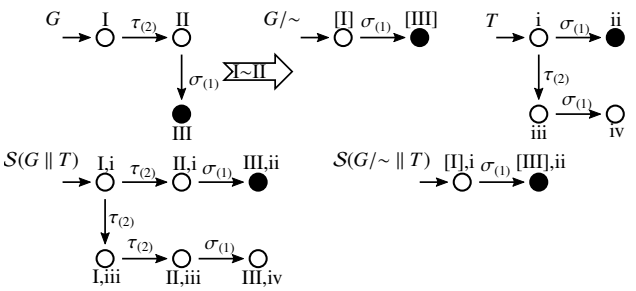

Fig. 2. a silent step with priority lower then its delayed non-silent action may not be mergable

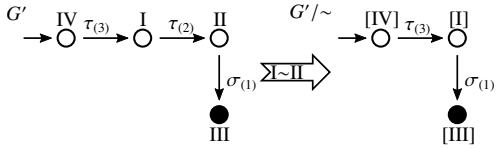

Fig. 3. redundant silent step rule

then $\mathsf{prio}(\sigma)$. The importance of this restriction for conflict equivalence can be seen from the following example. For the brevity in figures, we write the priority of each event in the subscript of transition labels and always assume that $\mathsf{prio}(\omega) = 1$ in the current and subsequent section.

Example 12. Let $G = \langle Q, A, \rightarrow, Q^\circ \rangle$ be such as given in Figure 2 with $\Sigma = \{\sigma, \omega\}$. From (P1), I $\not\approx$ II. If I and II are merged through some equivalence relation $\sim$, the counterexample T as given in Figure 2 can witness that $G \not\simeq_{\mathcal{S}} (G/\!\sim)$. □

Consider the automaton G as given in Figure 2 again. The failure of the abstraction can be seen as being caused by the *reachable* state (I, i) in $\mathcal{S}(G \parallel T)$ as state i has the chance to execute τ whose priority is lower then σ since $\sigma \notin G_{\mathrm{rglr}}(\mathrm{I})$. Interestingly, adding further restriction on the automaton can render such "bad" states to be unreachable. As for G in Figure 2, we could switch the initial state to a new state IV and add a new transition $\mathrm{IV} \xrightarrow{\tau_{(3)}} \mathrm{I}$. For such an automaton G' as given in Figure 3, merging I and II yields a conflict preserving abstraction. The intuition behind this modification is that due to the new transition, (I, i) becomes unreachable. In this case, we say $\mathrm{I} \xrightarrow{\tau} \mathrm{II}$ is a *redundant silent step*.

Definition 13. Let $G = \langle Q, A, \rightarrow, Q^\circ \rangle$ be a Υ-shaped automaton. A transition $x \xrightarrow{\tau} y$ with $x, y \in Q$ and $\tau \in \Upsilon$ is a *redundant silent step* if this is the only transition outgoing from x, $x \notin Q^\circ$ and $z \xrightarrow{\alpha} x$ for any $z \in Q$ implies $\alpha \in \Upsilon$ and $\mathsf{prio}(\alpha) > \mathsf{prio}(\tau)$. An equivalence $\sim \subseteq Q \times Q$ on G is *induced by the transition* $x \xrightarrow{\alpha} y$ if $x \sim y$ and for all $z \notin \{x, y\}$, $[z]$ is a singleton class. □

Given a redundant silent step $x \xrightarrow{\tau} y$ with some higher-priority non-silent events active in y, x and y are never prioritised weak bisimilar. Interestingly, we can show that all states which cause such a defect are not reachable. From this we obtain the following abstraction rule.

Theorem 14. (redundant silent step rule). Given a Υ-shaped automaton $G = \langle Q, A, \rightarrow, Q^\circ \rangle$ and the equivalence $\sim \subseteq Q \times Q$ induced by a redundant silent step. It then holds that $G \simeq_{\mathcal{S}} (G/\!\sim)$. □

5. ABSTRACTION RULES BASED ON INCOMING EQUIVALENCE

For ordinary conflict-preserving abstraction without event priorities, Flordal and Malik (2009) introduce the *active events rule* and the *silent continuation rule* which are based on a pre-partition through *incoming equivalence*. The key property of incoming equivalence in the ordinary set-up is that, if there is a trace beginning with a non-silent event in the composition after abstraction, then a trace with the same non-silent events can be constructed in the original automaton as well. To achieve this property with event priorities, we use the following notion of *string preservation*.

Definition 15. Let $G = \langle Q_G, A_G, \rightarrow_G, Q_G^\circ\rangle$ be a Υ-shaped automaton. An equivalence $\sim \subseteq Q \times Q$ on G is *string-preserving* if for any arbitrary automaton $T = \langle Q_T, A_T, \rightarrow_T, Q_T^\circ\rangle$ and any trace

$$([x_{G0}], x_{T0}) \xrightarrow{\alpha_1}_{\mathcal{S}} ([x_{G1}], x_{T1}) \xrightarrow{\alpha_2}_{\mathcal{S}} \cdots \xrightarrow{\alpha_k}_{\mathcal{S}} ([x_{Gk}], x_{Tk})$$

in $\mathcal{S}(G/\sim \parallel T)$ where $k \geq 1$, $\alpha_1 \in (A_G \cup A_T) - \Upsilon$ and $\alpha_i \in A_G \cup A_T$ for all $i \in \{2, \cdots, k\}$, there exist $x'_{G0} \in [x_{G0}]$ and $x'_{Gk} \in [x_{Gk}]$ so that $(x'_{G0}, x_{T0}) \xRightarrow{\mathsf{p}(\alpha_1 \cdots \alpha_k)}_{\mathcal{S}} (x'_{Gk}, x_{Tk})$ in $\mathcal{S}(G \parallel T)$. □

To achieve string preservation, we first conveniently define some further notations for transitions.

Definition 16. Given a Υ-shaped automaton $G = \langle Q, A, \rightarrow, Q^\circ\rangle$, define the following extended transition relations:

(T3) $\underset{!}{\rightarrow} \subseteq Q \times \Upsilon \times Q$: $x \xrightarrow[!]{\tau} y$ if $x \xrightarrow{\tau} y$ and $G^{<\tau}_{\mathrm{rglr}}(x) = \emptyset$.

(T4) $\underset{!n}{\hookrightarrow} \subseteq Q \times \{\epsilon\} \times Q$: $x \xhookrightarrow[!n]{\epsilon} y$ if

(i) either $n = 1$ and $x \xRightarrow[\Sigma:1]{\epsilon} y$,

(ii) or $n \geq 2$, $x \xrightarrow[!]{\tau_1} \xrightarrow[!]{\tau_2} \cdots \xrightarrow[!]{\tau_k} y$, $k \geq 1$ and $\mathsf{lo}(\{\tau_1, \cdots, \tau_k\}) = n$. □

Transition relations introduced in Definition 16 are generally more restrictive than those in Definition 9 in that all non-final states are individually in shaped form. Note that we intentionally use the new transition symbol "$\hookrightarrow$" since when $n \geq 2$, we do *not* have $x \xhookrightarrow[!n]{\epsilon} x$ for all x as at least one τ transition with $\mathsf{prio}(\tau) = n$ must happen during $\xhookrightarrow[!n]{\epsilon}$.

Based on Definition 16, we introduce the adapted incoming equivalence when considering priorities:

Definition 17. Let $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ be a Υ-shaped automaton. An equivalence $\sim_{\mathrm{inc}} \subseteq Q \times Q$ on G is an *incoming equivalence* if for any $x, x' \in Q$ so that $x \sim_{\mathrm{inc}} x'$, it holds that

(I1) for all $\sigma \in A - \Upsilon$, $n \geq 0$ and $y \in Q$, $y \xRightarrow[\Delta:\sigma]{\epsilon} \xrightarrow[\Delta:\sigma]{\sigma} \xhookrightarrow[!n]{\epsilon} x \Leftrightarrow y \xRightarrow[\Delta:\sigma]{\epsilon} \xrightarrow[\Delta:\sigma]{\sigma} \xhookrightarrow[!n]{\epsilon} x'$ where $\Delta = G^{<\sigma}_{\mathrm{rglr}}(y)$;

(I2) for any $n \geq 0$, $Q^\circ \xhookrightarrow[!n]{\epsilon} x \Leftrightarrow Q^\circ \xhookrightarrow[!n]{\epsilon} x'$;

(I3) for any $y \in Q$ and $\tau \in \Upsilon$, $y \xrightarrow{\tau} \xRightarrow{\epsilon} x$ or $y \xrightarrow{\tau} \xRightarrow{\epsilon} x'$ implies $G^{<\tau}_{\mathrm{rglr}}(y) = \emptyset$. □

Similar to the original version in Flordal and Malik (2009), Definition 17 attempts to equalise states which can be reached in the same way, i.e. we only care about the past of a state and ignore its future behaviour. However, such intuition is insufficient when event priorities are considered since string preservation requires the same state x_{T0} and x_{Tk} from some test T to be connected before and after abstraction. Without restrictions over the future behaviour of incoming equivalent states, string preservation can be easily invalidated when two equivalent states have different preemption power. In this regard, we introduce an adapted notion of active-event equivalence and silent-continuation equivalence.

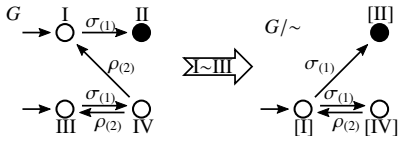

Fig. 4. active events rule

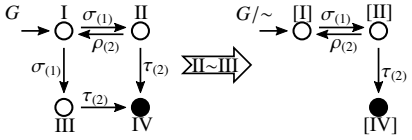

Fig. 5. silent continuation rule

Definition 18. Let $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ be a Υ-shaped automaton. An equivalence $\sim_{\mathrm{ae}} \subseteq Q \times Q$ on G is an *active-event equivalence* if for any $x, x' \in Q$ so that $x \sim_{\mathrm{ae}} x'$, either $x = x'$ or:

(AE1) $G_{\mathrm{slnt}}(x) = G_{\mathrm{slnt}}(x') = \emptyset$;

(AE2) $G_{\mathrm{rglr}}(x) = G_{\mathrm{rglr}}(x')$. □

Definition 19. Let $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ be a Υ-shaped automaton. An equivalence $\sim_{\mathrm{sc}} \subseteq Q \times Q$ on G is a *silent-continuation equivalence* if for any $x, x' \in Q$ so that $x \sim_{\mathrm{sc}} x'$, either $x = x'$, or there exists some $\tau \in \Upsilon$ with:

(SC1) $\tau \in G_{\mathrm{slnt}}(x) \cap G_{\mathrm{slnt}}(x')$;

(SC2) $G^{<\tau}_{\mathrm{rglr}}(x) = G^{<\tau}_{\mathrm{rglr}}(x') = \emptyset$;

(SC3) Neither x nor x' is in any live-lock. □

We are now in the position to state two more conflict equivalent abstraction rules.

Theorem 20. (active events rule). Let $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ be a Υ-shaped automaton with an equivalence $\sim \subseteq Q \times Q$ so that $\sim \subseteq \sim_{\mathrm{inc}} \cap \sim_{\mathrm{ae}}$. It holds that $G \simeq_{\mathcal{S}} (G/\sim)$. □

Example 21. Let $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ be such as given in Figure 4 with $\Sigma = \{\sigma, \rho, \omega\}$. I and III are incoming equivalent and $\sigma \in G(\mathrm{I}) \cap G(\mathrm{III})$. This indicates that they qualify the active events rule. □

Theorem 22. (silent cont. rule). Let $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ be a Υ-shaped automaton with an equivalence $\sim \subseteq Q \times Q$ so that $\sim \subseteq \sim_{\mathrm{inc}} \cap \sim_{\mathrm{sc}}$. It holds that $G \simeq_{\mathcal{S}} (G/\sim)$. □

Example 23. Let $G = \langle Q, A, \rightarrow, Q^\circ\rangle$ be such as given in Figure 5 with $\Sigma = \{\sigma, \omega\}$. States II and III are incoming equivalent and both can execute $\tau_{(2)}$. In addition, no events with higher priority are active in either state, indicating that II and III qualify for the silent continuation rule. □

While the two previous abstraction rules require a careful adaptation to event priorities, the following two carry over immediately from Flordal and Malik (2009).

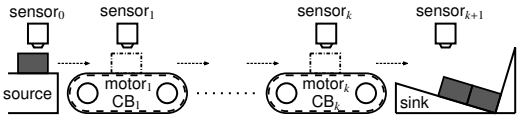

Fig. 6. Concatenated conveyor belts

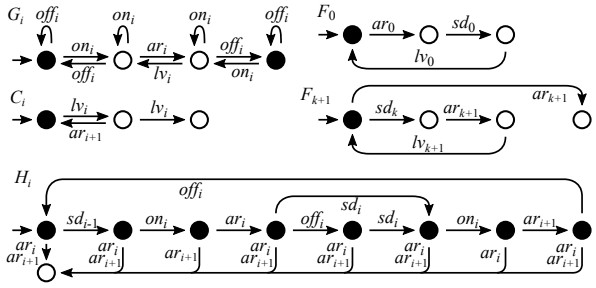

Fig. 7. Automata models for the conveyor belts example

Theorem 24. (only silent incoming rule). Let $G = \langle Q, A, \rightarrow, Q^\circ \rangle$ be a Υ-automaton and let $\bar{x} \in Q$ be such that $\bar{x}$ is not in any live-lock, $\bar{x} \xrightarrow{\tau_{(1)}}$ and $y \xrightarrow{\alpha} \bar{x}$ implies $\alpha = \tau_{(1)}$. Then for the automaton $H = \langle Q, A, \rightarrow_H, Q^\circ \rangle$ with

$$\rightarrow_H = \{(x, \alpha, y) \mid x \xrightarrow{\alpha} y \text{ and } y \neq \bar{x}\} \cup \{(x, \alpha, y) \mid x \xrightarrow{\tau_{(1)}} \bar{x} \xrightarrow{\alpha} y\}, \quad (3)$$

it holds that $G \simeq_{\mathcal{S}} H$. □

Theorem 25. (only silent outgoing rule). Let $G = \langle Q, A, \rightarrow, Q^\circ \rangle$ be a Υ-shaped automaton and let $\bar{x} \in Q$ be such that $\bar{x}$ is not in any live-lock, $G(\bar{x}) = \{\tau_{(1)}\}$ and $z \xrightarrow{\alpha'} \bar{x}$ implies $\alpha' \notin \Upsilon$. Let $\bar{Q} := \{y \in Q \mid \bar{x} \xrightarrow{\tau_{(1)}} y\}$, then for the automaton $H = \langle Q - \{\bar{x}\}, A, \rightarrow_H, Q^\circ_H \rangle$ with

$$Q^\circ_H = \begin{cases} Q^\circ & \text{if } \bar{x} \notin Q^\circ \\ (Q^\circ - \{\bar{x}\}) \cup \bar{Q} & \text{if } \bar{x} \in Q^\circ \end{cases} \quad (4)$$

$$\rightarrow_H = \{(x, \alpha, y) \mid x \xrightarrow{\alpha} y \text{ and } x \neq \bar{x} \text{ and } y \neq \bar{x}\} \cup \{(x, \alpha, y) \mid x \xrightarrow{\alpha} \bar{x} \text{ and } y \in \bar{Q}\}, \quad (5)$$

it holds that $G \simeq_{\mathcal{S}} H$. □

6. EXAMPLES

For our first example, consider the concatenated conveyor belts CB_1 to CB_k shown in Figure 6. In this scenario, workpieces are to be transported from the source on the left to the sink on the right. Each of the components is equipped with a sensor to indicate the presence of a workpiece. Besides, each conveyor belt is equipped with a motor which drives the belt. The components are controlled in a modular fashion, with the respective automata given in Figure 7; see also Table 1 for a listing of all referenced events. Specifically, each conveyor belt CB_i is modelled as a local closed loop $F_i := G_i \parallel H_i$, with the special cases F_0 and F_{k+1} for source and sink, and with successive components F_i and F_{i+1} coupled by the automaton C_i. The overall model so far is given by $F := \|_{0 \leq i \leq k}(F_i \parallel C_i) \parallel F_{k+1}$.

For a physical implementation of the controller, we effectively implement the behaviour of F with specific execution preferences. Actuators on_i and off_i are assigned a higher priority than the sensor events ar_i and lv_i; i.e., when in a state where the controller could either wait for a sensor event to occur or execute an actuator event at some point in time, the physical implementation of the controller will do the latter immediately. Likewise, the events sd_i for inter-module communication are preferred over actuator events. Only when in a state where exclusively sensor events are enabled, the physical implementation will wait until one such event is generated by the plant. Although intuitive from a technological perspective, this scheme of execution preferences may render the overall model blocking even it was non-blocking before introducing priorities. Hence our interest in the verification of $\mathcal{S}(F)$.

Table 1. Events in the conveyor belts example

event	description	priority
on_i	motor_i on	2
off_i	motor_i off	2
ar_i	sensor_i workpiece arrival	3
lv_i	sensor_i workpiece departure	3
sd_i	send workpiece from component i	1
ω	termination event	3

The performance of a prototypical software implementation of the abstraction rules discussed in our study is given in Table 2. The first column shows the number of conveyor belts. The second and third column show the state count of the monolithic representation and the final state count after applying compositional verification. The fourth and fifth column show the elapsed time for verification with or without applying compositional verification. All computations are performed on a standard 2022 desktop computer (Intel Core i7-10510U 2.30 GHz CPU with 16GB RAM). Through comparing the fourth and fifth column, a substantial performance improvement becomes evident.

Table 2. State count and elapsed time in sec

k	mono. st. cnt.	comp. st. cnt.	mono. time	comp. time
5	3.4×10^3	31	0.28	0.08
6	9.9×10^3	36	0.81	0.11
7	2.8×10^4	41	2.33	0.16
8	7.7×10^4	46	6.86	0.21
9	2.1×10^5	51	21.37	0.29
10	5.6×10^5	56	61.63	0.34

For our second example, we consider the scenario shown in Figure 9, consisting of two stack feeders ($\mathsf{SF}_{1/2}$), two conveyor belts ($\mathsf{CB}_{1/2}$), a processing machine (PM), a rotary table (RB) and an exit slide (XS). The intended behaviour is to forward workpieces from both stack feeders to the exit slide, where those that pass the processing machine shall be processed accordingly. While our first example demonstrates scalability using models on a high level of abstraction, our second example is a real-world case study which is run by a simulator on a physical level. To achieve the intended behaviour, the simulation can connect to an actual PLC, for which the specification is given by the SBDs in Figure 8. For the purpose of verification, the SBDs are converted to automata. Since firing transitions in an SBD by definition is of a higher priority than setting or clearing boolean variables of line levels, we are interested in non-conflictingness after application of the shaping operator. For the example at hand, a monolithic representation has state count of 4.3×10^4 which takes

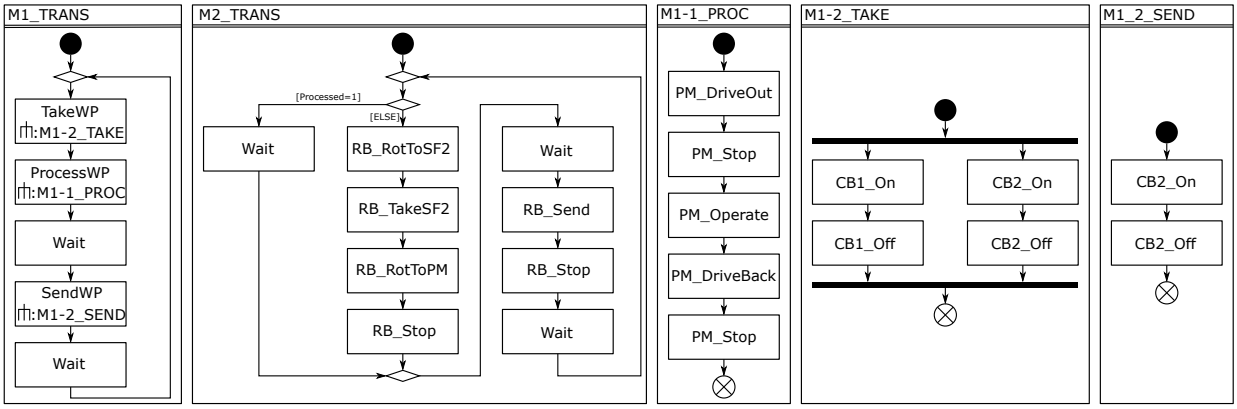

Fig. 8. Production line specification (SBDs)

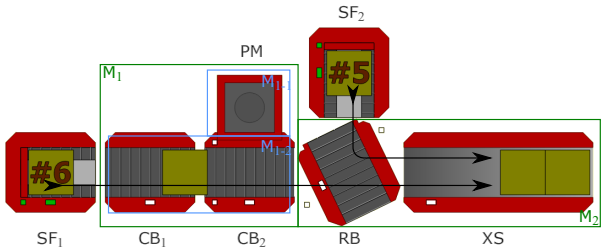

Fig. 9. Production line (physical simulation)

2.13s to construct and verify. Using the compositional approach discussed in this paper, the final state count amounts to 598 and the entire procedure takes 0.92s.

CONCLUSION

Considering a class of modular discrete-event systems with event priorities, we have presented a number of conflict equivalent abstractions to be used for compositional verification of non-blockingness. Technically, our study builds on Flordal and Malik (2009) and we inspect the abstraction rules presented there in order to derive adaptations that address priorities. This approach is closely related to the development of CCS^{ch} by Lüttgen (1998) which introduces priorities to process algebra CCS (Milner (1989)).

REFERENCES

Bauer, N., Huuck, R., Lukoschus, B., and Engell, S. (2004). A unifying semantics for sequential function charts. volume 3147, 400–418.

Brecher, C., Obdenbusch, M., Özdemir, D., Flender, J., Weber, A.R., Jordan, L., and Witte, M. (2016). Interdisciplinary specification of functional structurres for machine design. *IEEE International Symposium on Systems Engineering (ISSE)*.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer, second edition.

Cleaveland, R., Lüttgen, G., and Natarajan, V. (2007). Priority and abstraction in process algebra. *Information and Computation*, 205(9), 1426–1458.

Eshuis, R. (2007). Symbolic model checking of UML activity diagrams. *ACM Transactions on Software Engineering and Methodology (TOSEM)*.

Fabian, M. and Hellgren, A. (1998). PLC-based implementation of supervisory control for discrete event systems. *Proceedings of the 37th IEEE Conference on Decision and Control*.

Flordal, H. and Malik, R. (2009). Compositional verification in supervisory control. *SIAM J. Control and Optimization*, 48, 1914–1938.

LRT/FGDES (2019). CompileDES: Executable Code Generation from Synchronised libFAUDES Automata. `http://fgdes.tf.fau.de/compiledes`. Accessed: 22.03.2022.

Lüttgen, G. (1998). Pre-emptive modeling of concurrent and distributed systems.

Malik, R., Streader, D., and Reeves, S. (2004). Fair testing revisited: A process-algebraic characterisation of conflicts. In F. Wang (ed.), *Automated Technology for Verification and Analysis*, 120–134. Springer.

Milner, R. (1989). *Communication and Concurrency*. Prentice-Hall, Inc., USA.

Mohajerani, S., Malik, R., and Fabian, M. (2014). A framework for compositional synthesis of modular nonblocking supervisors. *IEEE Transactions on Automatic Control*, 59(1), 150–162.

Nicola, R.D. and Hennessy, M. (1984). Testing equivalences for processes. *Theoretical Computer Science*, 34(1), 83 – 133.

Pilbrow, C. and Malik, R. (2015). An algorithm for compositional nonblocking verification using special events. *Science of Computer Programming*, 113, 119–148.

Qamsane, Y., Abdelouahed, T., and Philippot, A. (2016). A synthesis approach to distributed supervisory control design for manufacturing systems with grafcet implementation. *International Journal of Production Research*, 55, 1–21.

Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event systems. *SIAM Journal on Control and Optimization*, 25, 206–230.

Su, R., van Schuppen, J.H., Rooda, J.E., and Hofkamp, A.T. (2010). Nonconflict check by using sequential automaton abstractions based on weak observation equivalence. *Automatica*, 46(6), 968 – 978.

Tang, Y. and Moor, T. (2022). Technical details regarding compositional verification with event-priorities. Report available at https://fgdes.tf.fau.de/publications.html (accessed July 16th 2022).

Ware, S. and Malik, R. (2012). Conflict-preserving abstraction of discrete event systems using annotated automata. *Discrete Event Dynamic Systems*, 22, 451–477.