Compositional Verification of Finite Automata under Event Preemption

Yiheng Tang[1] and Thomas Moor[2]

Abstract— **Given a number of synchronised automata, compositional verification seeks to verify non-conflictingness without the explicit computation of an overall model. Technically, the approach alternates conflict-preserving abstractions with the composition of a small number of strategically chosen automata and the literature reports substantial computational benefits for examples of practical relevance. In this paper, we re-visit this approach in order to address the situation of *preemptive events*, i.e., events that are known beforehand to be scheduled at highest priority. Our study is motivated by high-level programming languages commonly used in industrial automation, where events associated with actuators preempt events associated with sensors.**

I. Introduction

Modular and/or hierarchical architectures are an established approach for the design of large-scale dynamic systems. In the present paper, we focus attention to modular discrete-event systems (DES), i.e., the overall system is composed by individual modules, each being a DES representable by a finite automaton. In particular, each module refers to a specific alphabet and the composition of the overall system is via synchronisation of shared events [1]. For this system class the overall state count of a monolithic representation is exponential in the number of modules. Given a specific analysis or synthesis task, the question hence is whether and how it can be addressed while avoiding an explicit construction of a monolithic representation. This highly relevant question has been addressed extensively in the recent literature; for supervisory controller synthesis, see e.g. [2], [3], [4], for analysis see e.g. [5], [6], [7], [8], [9]. In the present paper, we address the verification of non-conflictingness and we do so by following the approach of *compositional verification*, as originally introduced by [6]. The basic plot is to (i) simplify individual modules such that conflicts in the overall system are retained (ii) strategically compose a small number of modules (iii) iterate the former two steps until there is only one module left. There is a great variety of so called *conflict-preserving abstractions* that qualify for step (i); for plain finite state machines see again [6], [7], [8], [9], for extended automata see e.g. [10], [11]. Other well discussed properties of DES under composition, e.g. controllability [6] and opacity [12], were also shown being transformable into the compositional non-blocking verification. Recently, [13] also showed that the compositional verification approach can be applied to efficiently verify any temporal logical properties within CTL*-X.

[1,2]Lehrstuhl für Regelungstechnik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstr. 7, 91058 Erlangen, Germany; Email lrt@fau.de

In our study, we include a distinguished class of events in order to address the practical situation where individual modules represent the behaviour of a programmable logic controller (PLC) that interacts with a physical plant. In such a setting, certain events will represent edges on line levels associated with actuators, i.e., TurnMotorOn to activate a motor of a conveyor belt in an automated manufacturing system. When considering a state in which multiple events are enabled, events associated with an actuator differ semantically from the remaining events: the PLC will instantly activate one of the eligible actuators and proceed to the successor state. Hence, we would like to explicitly distinguish *preemptive* events in contrast to the remaining events they preempt. This notion of preemption is also common in models derived from Statecharts [14], like hierarchical finite state machines in Ptolomy II [15], where outgoing transitions of a so called *macrostate* can be preemptive; or some Petri-net derivatives, e.g., Grafcet as defined by the IEC-60848 [16], and UML Activity Diagrams [17], [18], where token reconfiguration is considered as preemptive. A convenient way to organise verification tasks for such DES is to translate their behaviour into finite automata models at a pre-processing stage which inevitably inherits the notion of preemptive events and this motivates our study. For related work, that addresses abstraction techniques in with event preemption from a process-algebraic perspective, see e.g. [19], [20], [21], [22].

If there was only one module, we would *shape* the automaton by removing any outgoing transitions labelled by non-preemptive events if at least one preemptive event is active. However, for the case of multiple modules it is readily observed that the proposed shaping does in general not commute with synchronous composition. Hence we need to re-visit conflict-preserving abstractions for our specific use-case. As our main contribution, we identify a number of abstraction methods that are *conflict-preserving under preemption* and that comply with the proposed semantics of preemptive events.

Remark 1. We are aware of the fact that our notion of preemption is covered by the more general setting of timed DES (TDES); see e.g. [23]. However, rather than to attempt to lift the compositional approach to a TDES setting, we believe that it is conceptually more promising to build on as many readily available results as possible by only considering our very specific notion of preemption. □

This paper is organised as follows. After recalling notation and preliminaries in Section II, we discuss various methods

of conflict-preserving abstraction in Section III. The overall procedure of compositional verification is then presented in Section IV. We demonstrate computational benefits of our approach by a simple but yet practical example in Section V. Formal proofs are provided via the Appendix.

II. Preliminaries

A. Events and languages

An *alphabet* Σ is a finite set of *symbols* also referred to as *events*. Next to ordinary events, we consider three distinguished events. The two *silent events* τ^n and τ^p represent internal operations of individual modules; see [1]. The two variants differ semantically regarding preemption, which we discuss in detail below; see Definition 4. The *termination event* ω indicates process termination; see also [6]. By convention, symbols that denote an alphabet do not contain any of the three distinguished events, i.e. $\{\tau^n, \tau^p, \omega\} \cap \Sigma = \emptyset$, and we explicitly write $\Sigma \cup \{\tau^n, \tau^p, \omega\}$ whenever we refer to a set of symbols including the three distinguished events.

A *string* is a finite sequence of events. The *empty string* is denoted ϵ. Referring to a range of symbols, the *Kleene closure* $(\cdot)^*$ denotes the set of all strings; e.g., Σ^* denotes the set of all finite sequences of events from Σ, including the empty string $\epsilon \in \Sigma^*$. For two strings s and t, the concatenation is denoted st. Note that $\epsilon s = s = s\epsilon$ for any string s. To annihilate any information regarding the occurrence of silent events encoded by a specific string, we consider the following three *natural projections*

$$\mathsf{p}^n : (\Sigma \cup \{\tau^n, \tau^p, \omega\})^* \to (\Sigma \cup \{\tau^p, \omega\})^*; \tag{1}$$

$$\mathsf{p}^p : (\Sigma \cup \{\tau^n, \tau^p, \omega\})^* \to (\Sigma \cup \{\tau^n, \omega\})^*; \tag{2}$$

$$\mathsf{p} : (\Sigma \cup \{\tau^n, \tau^p, \omega\})^* \to (\Sigma \cup \{\omega\})^* \tag{3}$$

which delete all τ^n, or all τ^p, or all τ^n and all τ^p events, respectively; see e.g. [24] for a formal definition.

B. Automata

Definition 1. A *finite automaton* is a tuple $G = \langle Q, \Sigma, \to, Q^0 \rangle$ with Q a finite *state set*, Σ a finite *alphabet*, $\to \subseteq Q \times (\Sigma \cup \{\tau^n, \tau^p, \omega\}) \times Q$ the transition relation, and $Q^0 \subseteq Q$ a set of *initial states*. □

Note that we do not require automata to be deterministic. We use the infix notation $x \xrightarrow{\sigma} y$ for $(x, \sigma, y) \in \to$. The event σ is *active in state* x if there exists y such that $x \xrightarrow{\sigma} y$. This is denoted $x \xrightarrow{\sigma}$.

The transition relation is extended to string-valued labels in the common way; i.e., (i) let $x \xrightarrow{\epsilon} x$ for all $x \in Q$ and (ii) iteratively let $x \xrightarrow{s\sigma} z$ for all $x, z \in Q$, $s \in (\Sigma \cup \{\tau^n, \tau^p, \omega\})^*$ and $\sigma \in \Sigma \cup \{\tau^n, \tau^p, \omega\}$ provided that $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$. The set-theoretic complement of the extended transition relation is denoted $\not\to$, i.e., we write $x \xrightarrow{s} \!\!\!\!\!\!/\;\; y$ for $(x, s, y) \notin \to$. Moreover, we write $X \xrightarrow{s} Y$ for $X, Y \subseteq Q$ whenever there exist $x \in X$ and $y \in Y$ s.t. $x \xrightarrow{s} y$. Finally, $X \xrightarrow{s}$ and $G \xrightarrow{s}$ stand for $X \xrightarrow{s} Q$ and $Q^0 \xrightarrow{s} Q$, respectively.

As with natural projections on strings, we associate with $\to \subseteq Q \times (\Sigma \cup \{\tau^n, \tau^p, \omega\})^* \times Q$ the *abstract transition relation* $\Rightarrow \subseteq Q \times (\Sigma \cup \{\omega\})^* \times Q$ to annihilate any requirements imposed on the occurrence of silent events. Technically, we define $x \overset{s}{\Rightarrow} y$ if and only if $x \xrightarrow{s'} y$ for some $s' \in (\Sigma \cup \{\tau^n, \tau^p, \omega\})^*$ s.t. $\mathsf{p}(s') = s$. The abstract transition relations $\Rightarrow^n \subseteq Q \times (\Sigma \cup \{\tau^p, \omega\})^* \times Q$ and $\Rightarrow^p \subseteq Q \times (\Sigma \cup \{\tau^n, \omega\})^* \times Q$ are defined literally the same, except we now refer to the projections p^n and p^p, respectively.

The termination event ω indicates process termination; i.e., there will be no events after ω. For a well formed automaton, we have that $x \xrightarrow{\omega} y$ implies $y \xrightarrow{\sigma} \!\!\!\!\!\!/\;\;$ for all $\sigma \in \Sigma \cup \{\tau^n, \tau^p, \omega\}$. In this case, x is a *marked state* while y is a *terminal state*. For graphical representations, marked states are denoted by full black shapes and terminal states are omitted for clarity.

Given two automata, their synchronous behaviour is represented by their *synchronous composition* [1].

Definition 2. Given automata $G_1 = \langle Q_1, \Sigma_1, \to_1, Q_1^0 \rangle$ and $G_2 = \langle Q_2, \Sigma_2, \to_2, Q_2^0 \rangle$, their synchronous composition is defined by

$$G_1 \parallel G_2 := \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \to, Q_1^0 \times Q_2^0 \rangle$$

where

$$(x_1, x_2) \xrightarrow{\sigma} (x_1', x_2') \text{ if } \sigma \in (\Sigma_1 \cap \Sigma_2) \cup \{\omega\},\ x_1 \xrightarrow{\sigma}_1 x_1',\ x_2 \xrightarrow{\sigma}_2 x_2';$$
$$(x_1, x_2) \xrightarrow{\sigma} (x_1', x_2) \text{ if } \sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau^n, \tau^p\},\ x_1 \xrightarrow{\sigma}_1 x_1';$$
$$(x_1, x_2) \xrightarrow{\sigma} (x_1, x_2') \text{ if } \sigma \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau^n, \tau^p\},\ x_2 \xrightarrow{\sigma}_2 x_2'. \qquad \square$$

For multiple synchronised automata, we say an event is *shared* if it appears in the alphabet of more than one automata, otherwise it is *private*.

A common approach to automata abstraction are *quotient automata*, i.e., one considers a suitable equivalence relation $\sim \subseteq Q \times Q$ and merges equivalent states. We denote $[q] := \{q' \in Q \mid (q, q') \in \sim\}$ the *equivalence class* associated with a state $q \in Q$ w.r.t. $\sim$ for the following formal definition.

Definition 3. Given an automaton $G = \langle Q, \Sigma, \to, Q^0 \rangle$ and an equivalence relation $\sim \subseteq Q \times Q$, the *quotient automaton* $G/\!\sim$ of G w.r.t. $\sim$ is defined by $G/\!\sim := \langle Q/\!\sim, \Sigma, \to/\!\sim, \tilde{Q}^0 \rangle$ where $Q/\!\sim = \{[q] \mid q \in Q\}$, $\to/\!\sim = \{[p] \xrightarrow{\sigma} [q] \mid p \xrightarrow{\sigma} q\}$ and $\tilde{Q}^0 = \{[q^0] \mid q^0 \in Q^0\}$. □

C. Event preemption

We distinguish *preemptive events* in contrast to the remaining *non-preemptive* events. Whenever an automaton attains some state in which at least one preemptive event is active, some active preemptive event must occur next. This effectively disables the non-preemptive active events and is expressed by the following *shaping* operation.

Definition 4. Given an automaton $G = \langle Q, \Sigma, \to, Q^0 \rangle$ and a set of *preemptive events* $\Sigma^p \subseteq \Sigma \cup \{\omega\}$, denote

$$Q^p := \{\, x \in Q \mid \exists\, \sigma^p \in \Sigma^p \cup \{\tau^p\} :\ x \xrightarrow{\sigma^p} \,\} \tag{4}$$

the set of *preemptive states*. Then the *shaping operator* $\mathcal{S}$ is defined by $\mathcal{S}(G) := \langle Q, \Sigma, \to^{\mathcal{S}}, Q^0 \rangle$ where $p \xrightarrow{\sigma}{}^{\mathcal{S}} q$ if and only if $p \xrightarrow{\sigma} q$ and (i) $\sigma \in \Sigma^p \cup \{\tau^p\}$ or (ii) $p \notin Q^p$. □

Preemptive states are states with some active preemptive event. Note that $\rightarrow^{\mathcal{S}} \subseteq \rightarrow$ and clause (i) ensures that preemptive events remain active, while clause (ii) ensures that non-preemptive events remain active provided that no preemptive event is active. Note that we optionally allow for termination ω to be preemptive. Moreover, the above definition always treats $\tau^p \notin \Sigma$ as preemptive and $\tau^n \notin \Sigma$ as non-preemptive. We say an automaton G is *shaped* if $G = \mathcal{S}(G)$.

The shaping operator in general does not commute with the synchronous composition. Considering two automata G_1 and G_2 that share some preemptive events, we may encounter that a state (x_1, x_2) in $G_1 \parallel G_2$ is not preemptive while x_1 and/or x_2 are preemptive states in G_1 or G_2, respectively.

III. Conflict Preserving Abstraction w.r.t. Preemption

An automaton is non-blocking if from all reachable states in some future ω can be executed, i.e., if faithful termination is persistently possible. For synchronised automata, the terminology of non-conflictingness is usually used synonymously for non-blockingness of the synchronous composition. This is not adequate when considering event preemption. Here, we are interested in a non-blocking overall behaviour *after shaping*.

Definition 5. An automaton $G = \langle Q, \Sigma, \rightarrow, Q^0 \rangle$ is *non-blocking* if for each $x \in Q$ s.t. $G \overset{s}{\Rightarrow} x$ for some $s \in \Sigma^*$, there exists $t \in \Sigma^*$ s.t. $x \overset{t\omega}{\Longrightarrow}$. A family $(G_i)_{1 \leq i \leq n}$ of automata is *non-conflicting w.r.t. preemption* if $\mathcal{S}(G_1 \parallel G_2 \parallel \cdots \parallel G_n)$ is non-blocking. □

Suppose we are given a family of automata and ask whether they are non-conflicting w.r.t. preemption. This question can be conventionally answered by first computing an explicit representation of the synchronous composition, shaping the result, and finally testing for non-blockingness. However, on a relevant scale we must expect an intractable state count for the intermediate result, which motivates us to seek proper abstraction methods that we can apply to individual automata while not affecting the final result. However, since shaping in general does not commute with the synchronous composition, we need to adapt the known results to account for our specific situation of preemptive events. To this end, we impose the following requirement.

Definition 6. Given two automata G and G', we say they are *conflict equivalent w.r.t. preemption*, denoted by $G \simeq_{\mathcal{S}} G'$, if for any automaton T, it holds that G and T are non-conflicting w.r.t. preemption if and only if G' and T are non-conflicting w.r.t. preemption. □

For a family of automata $G_1, \cdots, G_n$ we may replace any one G_i, $1 \leq i \leq n$ by an abstraction G_i'. As long as the abstraction is conflict equivalent w.r.t. preemption, checking $\mathcal{S}(G_1 \parallel G_2 \parallel \cdots \parallel G_i' \parallel \cdots \parallel G_n)$ for non-blockingness will lead to the same results as checking $\mathcal{S}(G_1 \parallel G_2 \parallel \cdots \parallel G_n)$. This argument readily extends to abstracting multiple component automata and/or to composing a small number of automata to increase the chances to find adequate abstractions.

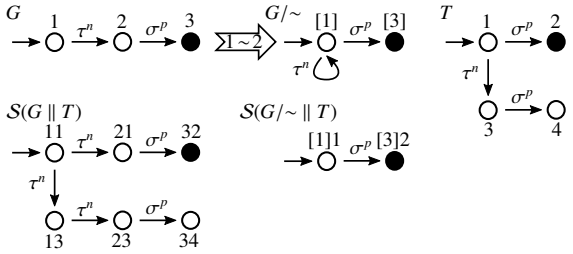

Fig. 1. Abstraction through observation equivalence is not conflict-preserving w.r.t. preemption (the τ^n self-loop in $G/\sim$ is preserved due to the definition of quotient automaton)

Our candidates for abstraction are quotient automata and we are left to impose suitable conditions on equivalence relations that lead to conflict equivalent abstractions w.r.t. preemption. One such condition is the following.

Definition 7. Let $G = \langle Q_G, \Sigma_G, \rightarrow_G, Q_G^0 \rangle$ be an automaton with an equivalence relation $\sim \subseteq Q_G \times Q_G$. Then $\sim$ is *reachability consistent on* G if for any automaton $T = \langle Q_T, \Sigma_T, \rightarrow_T, Q_T^0 \rangle$ and any event $\sigma \in \Sigma_G \cup \Sigma_T \cup \{\tau^n, \tau^p, \omega\}$, it holds

(C1) if $([x_G], x_T) \overset{\sigma}{\rightarrow}^{\mathcal{S}} ([y_G], y_T)$, then for any $x_G' \in [x_G]$, there exists $y_G' \in [y_G]$ s.t. $(x_G', x_T) \overset{\mathsf{p}(\sigma)}{\Longrightarrow}^{\mathcal{S}} (y_G', y_T)$;

(C2) if $(x_G, x_T) \overset{\sigma}{\rightarrow}^{\mathcal{S}} (y_G, y_T)$, then $([x_G], x_T) \overset{\sigma}{\rightarrow}^{\mathcal{S}} ([y_G], y_T)$. □

The above definition is a variation of the well-known observation equivalence [1], with the main difference that we now explicitly refer to the test automaton T. To appreciate the necessity of our variation, consider the example in Fig. 1. The two automata G and T are given with alphabets $\Sigma_G = \Sigma_T = \Sigma_G^p = \Sigma_T^p = \{\sigma^p\}$ and the global behaviour $\mathcal{S}(G \parallel T)$ is blocking. However, merging the two observation equivalent states 1 and 2 in G leads to a non-blocking global behaviour $\mathcal{S}(G/\sim \parallel T)$. Thus, observation equivalence is generally not suitable for our purposes. In contrast, reachability consistence is guaranteed to lead to adequate abstractions.

Proposition 1. Let $G = \langle Q_G, \Sigma_G, \rightarrow_G, Q_G^0 \rangle$ be an automaton with a reachability consistent equivalence relation $\sim \subseteq Q_G \times Q_G$ on G. It then holds that $G \simeq_{\mathcal{S}} G/\sim$. □

For practical purposes it now remains to find reachability consistent equivalence relations.

A. Abstract non-preemptive silent event

In this section, we focus attention on abstractions through projecting the non-preemptive silent event τ^n. We define a so called *observation equivalence w.r.t. non-preemptive events (OEn)*, which is finer than observation equivalence and is reachability consistent on any *shaped* automaton.

Definition 8. Given an automaton $G = \langle Q, \Sigma, \rightarrow, Q^0 \rangle$, an equivalence relation $\sim^n \subseteq Q \times Q$ is an *observation equivalence w.r.t. non-preemptive events* (OEn) on G if for all states

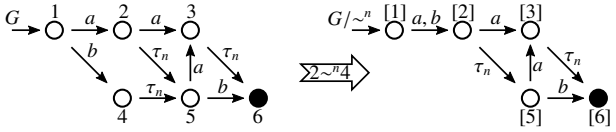

Fig. 2. Abstraction example through OEn

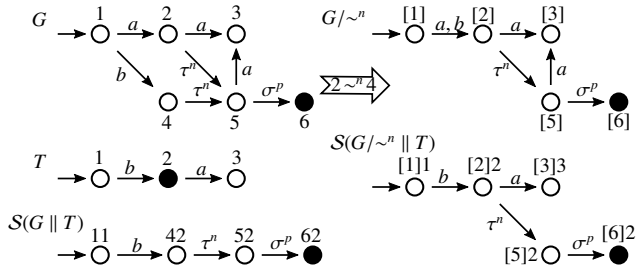

Fig. 3. Counterexample for abstraction being not conflict-preserving w.r.t. preemption when utilising OEn on an unshaped automaton

$x_1, x_2 \in Q$ s.t. $x_1 \sim^n x_2$ and for all $\sigma \in \Sigma \cup \{\tau^n, \tau^p, \omega\}$, the following holds:

(N1) if $x_1 \xrightarrow{\sigma} y_1$ for some $y_1 \in Q$, then there exists $\tilde{x}_2, y_2 \in Q$ s.t. $x_2 \overset{\epsilon}{\Rightarrow}^n \tilde{x}_2 \xrightarrow{\mathsf{p}^n(\sigma)} y_2$ and $y_1 \sim^n y_2$;

(N2) if $x_1 \in Q^p$ then $x_2 \in Q^p$. □

Proposition 2. Given a shaped automaton $G = \langle Q, \Sigma, \rightarrow, Q^0 \rangle = \mathcal{S}(G)$ with an OEn $\sim^n \subseteq Q \times Q$, it holds that $G \simeq_{\mathcal{S}} G/\sim^n$. □

An example of abstraction through OEn is given in Fig. 2. $G = \langle Q, \Sigma, \rightarrow, Q^0 \rangle$ is shaped with $\Sigma = \{a, b\}$ and $\Sigma^p = \{\omega\}$. An OEn $\sim^n \subseteq Q \times Q$ can be found s.t. $(2, 4) \in \sim^n$. Note that $(3, 6) \notin \sim^n$ since state 6 is preemptive but state 3 is not, thus they must remain distinguishable after abstraction.

Also note a counterexample given in Fig. 3 when the prerequisite of shaped input automata is not met: two automata G and T are given s.t. $\Sigma_G = \{a, b, \sigma^p\}$, $\Sigma_G^p = \{\sigma^p\}$, $\Sigma_T = \{a, b\}$, $\Sigma_T^p = \emptyset$, with $G \neq \mathcal{S}(G)$. The global behaviour $\mathcal{S}(G \parallel T)$ is non-blocking. Since state 52 is preemptive, state 33 is unreachable, but becomes reachable in $\mathcal{S}(G/\sim^n \parallel T)$ since states 2 and 4 are equivalent and thus are being merged, rendering $\mathcal{S}(G/\sim^n \parallel T)$ blocking.

To compute OEn, one can first pre-partitioning the state set of a given automaton into two classes, preemptive and non-preemptive states. Next, the abstract transition relation is constructed and the partition is refined to a bisimulation. In our implementation, we have applied the two-pass change-tracking algorithm for partitioning *delayed bisimulation* as introduced in [25]. By factoring τ^n-loops beforehand , the adopted algorithm uses topological sort of τ^n transitions to identify the delayed transition relation by a dynamic programming approach, as proposed by [26].

B. Abstract preemptive silent event

In this section, we focus attention on abstraction through projecting the preemptive silent event τ^p. We first define the observation equivalence w.r.t. preemptive events (OEp).

Definition 9. Given an automaton $G = \langle Q, \Sigma, \rightarrow, Q^0 \rangle$, an equivalence relation $\sim^p \subseteq Q \times Q$ is an *observation equivalence w.r.t. preemptive events* (OEp) on G if for all states $x_1, x_2 \in Q$ s.t. $x_1 \sim^p x_2$ and for all $\sigma \in \Sigma \cup \{\tau^n, \tau^p, \omega\}$, the following holds:

(P1) If $x_1 \xrightarrow{\sigma} y_1$ for some $y_1 \in Q$, then there exists $y_2 \in Q$ s.t. $x_2 \overset{\mathsf{p}^p(\sigma)}{\Longrightarrow}^p y_2$ and $y_1 \sim^p y_2$;

(P2) If $x_1 \in Q^p$ then $x_2 \in Q^p$. □

For any automaton, OEp implies (C1) of Definition 7. In comparison to OEn, matching (C1) for OEp is somewhat more "relaxed" then the case for OEn. Firstly, we can equivalently replace $x_2 \overset{\mathsf{p}^p(\sigma)}{\Longrightarrow}^p y_2$ by $x_2 \overset{\epsilon}{\Rightarrow}^p \tilde{x} \xrightarrow{\mathsf{p}^p(\sigma)} \tilde{y} \overset{\epsilon}{\Rightarrow}^p y_2$ in (P1) without changing the meaning of Definition 9 while analogically in (N1), after $\tilde{x} \xrightarrow{\mathsf{p}^n(\sigma)}$, we do not allow additional τ^n transitions to connect the target state y_2, since such τ^n transitions are either enforced to be postponed or even completely deactivated by preemptive transitions (even loops) with some test automaton. However, when discussing OEp, τ^p itself is preemptive and will not be shaped. Secondly, we do not require the input automaton to be in shaped form when considering OEp, since the part of "preemptive behaviour" will not be affected by the final shaping. However, (C2) can only be satisfied if the given automaton does not share any preemptive events with the test automaton.

Proposition 3. Given an automaton $G = \langle Q_G, \Sigma_G, \rightarrow_G, Q_G^0 \rangle$ with an OEp $\sim^p \subseteq Q_G \times Q_G$, it holds for any automaton $T = \langle Q_T, \Sigma_T, \rightarrow_T, Q_T^0 \rangle$ with $\Sigma_G^p \cap \Sigma_T^p = \emptyset$ that $\mathcal{S}(G \parallel T)$ is non-blocking if and only if $\mathcal{S}(G/\sim^p \parallel T)$ is non-blocking. □

When the prerequisite $\Sigma_G^p \cap \Sigma_T^p = \emptyset$ is not met, (C2) may be invalidated in that although OEp separates preemptive states from non-preemptive ones, it may happen that for two equivalent preemptive states, one becomes non-preemptive when synchronizing with some test, while the other is always preemptive by having τ^p active. This motivates us to refine OEp s.t. preemptive states with or without active τ^p are distinguished.

Definition 10. Given an automaton $G = \langle Q, \Sigma, \rightarrow, Q^0 \rangle$, an equivalence relation $\approx^p \subseteq Q \times Q$ is a *strong observation equivalence w.r.t. preemptive events* (SOEp) on G if for all states $x_1, x_2 \in Q$ s.t. $x_1 \approx^p x_2$ and for all $\sigma \in \Sigma \cup \{\tau^n, \tau^p, \omega\}$, the following holds:

(S1) If $x_1 \xrightarrow{\sigma} y_1$ for some $y_1 \in Q$, then there exists $y_2 \in Q$ s.t. $x_2 \overset{\mathsf{p}^p(\sigma)}{\Longrightarrow}^p y_2$ and $y_1 \approx^p y_2$;

(S2) If $x_1 \xrightarrow{\tau^p}$ then $x_2 \xrightarrow{\tau^p}$ □

Note that in SOEp, it is still guaranteed that preemptive states are never equivalent to non-preemptive states. More precisely, for any $x_1 \approx^p x_2$ and $x_1 \not\xrightarrow{\tau^p}$, x_1 and x_2 have exactly the same set of active events. It turns out that SOEp is reachability consistent on any automaton, which implies conflict equivalence w.r.t. preemption.

Proposition 4. Let $G = \langle Q, \Sigma, \rightarrow, Q^0 \rangle$ be an automaton with

a SOEn $\approx^p \subseteq Q \times Q$. It then holds that $G \simeq_{\mathcal{S}} G/\approx^p$. □

Similar to OEn, the computation of OEp as well as SOEp can be accomplished by the topological-sort based change-tracking algorithm for *weak bisimulation* introduced in [25], which is based on a topological sort of states w.r.t. τ^p-transitions. Note that states on a τ^p-loop are always OEp or SOEp equivalent, but the resulting τ^p-self-loop in the quotient cannot be removed unless another non-self-looping τ^p-transition is active. The topological sort shall indeed neglect those τ^p-self-loops.

IV. Compositional Verification

Recall that given for a family of automata G_i, $1 \leq i \leq n$, the global behaviour amounts to $G := \mathcal{S}(G_1 \parallel G_2 \parallel \cdots \parallel G_n)$, where each G_i is subject to abstractions by the methods introduced in Section III. However, when modelling each G_i from first principles, silent events are not directly utilised, and this potentially results in overly coarse state equivalences. The common procedure applied at this stage substitutes transitions with local events by τ-transitions. This is referred to as *hiding* [6], and we adapt the procedure to account for preemptive events.

Definition 11. Given an automaton $G = \langle Q, \Sigma, \rightarrow, Q^0 \rangle$ and an event set $\Delta \subseteq \Sigma$, define $\rightarrow^h$ by

$$x \xrightarrow{\tau^n}{}^h y \quad \text{if } x \xrightarrow{\sigma} y \text{ and } \sigma \in \Delta \setminus \Sigma^p; \tag{5}$$

$$x \xrightarrow{\tau^p}{}^h y \quad \text{if } x \xrightarrow{\sigma} y \text{ and } \sigma \in \Delta \cap \Sigma^p; \tag{6}$$

$$x \xrightarrow{\sigma}{}^h y \quad \text{if } x \xrightarrow{\sigma} y \text{ and } \sigma \notin \Delta\,. \tag{7}$$

Then *hiding* Δ *in* G results in the automaton $G \setminus \Delta := \langle Q, \Sigma \setminus \Delta, \rightarrow^h, Q^0 \rangle$. □

To abstract an automaton for compositional non-blocking verification, it is obvious that substituting all private events correspondingly into τ^n or τ^p will not affect non-blockingness of the global behaviour. After hiding private events Δ_i in each G_i, applying abstraction methods introduced in Section III on each $G_i \setminus \Delta_i$ will achieve a more appreciable state reduction. Afterwards, a small set of automata is chosen and replaced by their synchronous composition. Since this potentially introduces new private events, hiding and abstraction shall be performed again. These two computational steps are iterated until only one automaton remains. By construction, the non-blockingness of this finally shaped automaton coincides with that of the global behaviour. This iteration has been proposed in [9, Algorithm 1] with pseudo-code. In addition, if the verification result is negative, a counterexample (i.e. a trace from some initial state to some blocking state) can be directly computed through the state merging expansion algorithm introduced in [27].

We now comment on the detailed procedure of abstracting one individual automaton. Note that abstracting through OEn requires the input automaton to be in shaped form. In fact, we can always shape an automaton without influencing the global behaviour as long as all preemptive events are private.

Proposition 5. Let $G_1 = \langle Q_1, \Sigma_1, \rightarrow_1, Q_1^0 \rangle$ and $G_2 = \langle Q_2, \Sigma_2, \rightarrow_2, Q_2^0 \rangle$ be two automata s.t. $\Sigma_1^p \cap \Sigma_2^p = \emptyset$. It then holds that $\mathcal{S}(G_1 \parallel G_2) = \mathcal{S}(\mathcal{S}(G_1) \parallel G_2)$. □

The above proposition reads as follows: if some state is preemptive due to some private preemptive event, this state will remain preemptive under any composition since no other automaton can deactivate this event. Thus, Proposition 5 can be further relaxed in that we can always shape an automaton by only using its private preemptive events (shared preemptive events will not shape non-preemptive events, but not being shaped either). When hiding is properly performed, we can always shape an automaton w.r.t. τ^p, i.e. replace G_i by $\mathcal{S}_{\{\tau^p\}}(G_i)$ where $\mathcal{S}_{\{\tau^p\}}(\cdot)$ denotes a specific shaping operation which deactivates non-preemptive events only when τ_p is active on this state. To this end, each single automaton G_i can be abstracted s.t. we first hide its private events, shape it only w.r.t. τ_p and then decide abstraction methods to be applied based on whether it shares preemptive events or not:

1) if there are no shared preemptive events, construct a quotient automaton w.r.t. OEn and OEp since it always holds that $\mathcal{S}_{\{\tau^p\}}(G_i) = \mathcal{S}(G_i)$;
2) if shared preemptive events exist, check if the automaton is in shaped form and construct the quotient automaton w.r.t. OEn if possible. Then construct the quotient automaton w.r.t. SOEp.

Obviously, attempting to eliminate shared preemptive events while heuristically choosing automata to compose during the compositional non-blocking verification brings great advantage for state reduction since both OEn and OEp have more chances to be applied.

V. Example

Consider a transport system with k consecutive conveyor belts delivering workpieces from a source to a sink, see Fig. 4. Each conveyor belt as well as the source and the sink is equipped with a sensor detecting the arrival and departure of workpieces. Moreover, each conveyor belt is driven by a motor. The controlled behaviour of each conveyor belt CB_i is described by four automata, two for the uncontrolled plant behaviour (G_i^a and G_i^b) and two more for the modular

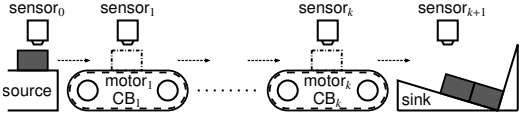

Fig. 4. A k-conveyor-belt example

TABLE I
Events in the transport-system example

event	description	preemptive
on_i	motor_i on	yes
off_i	motor_i off	yes
ar_i	sensor_i workpiece arrival	no
lv_i	sensor_i workpiece departure	no
in_i	CB_i workpiece entrance	no

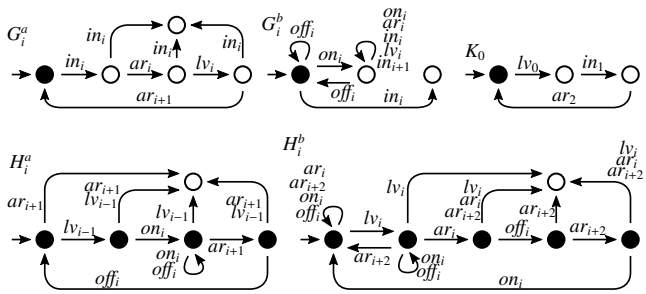

Fig. 5. Automata of the k-conveyor-belt example

controller (H_i^a and H_i^b), see Fig. 5. All utilised events are listed in Table I. In particular, on_i and off_i are associated with actuators which are intended to preempt all sensor events; i.e., for the global behaviour, actuator events can always be executed quickly enough without any delay that would allow for the occurrence of a sensor event. The automata for each CB_i in Fig. 5 are designed in the following manner:

a) Plant: The two models G_i^a and G_i^b are interpreted by synchronous composition. G_i^a describes the cyclic detection of workpieces while G_i^b implies that sensor events can only occur when the motor is on. As plant components, they formally allow for any actuator event in any state. However, to easy modelling, actuator events that are considered inappropriate lead to a dedicated error state, which is blocking. Special care is to be taken for in_i events: although they are associated with fictive sensors, we must declare by which component they are driven; from the perspective of CB_i, in_i is considered an actuator event imposed on CB_i, i.e. the preceding conveyor belt CB_{i-1} may potentially deliver a workpiece to CB_i at any time. In contrast, in_{i+1} semantically is a sensor controlled for CB_i, i.e. CB_i decides when to deliver a workpiece to the successive CB_{i+1}. Thus, in_i at specific states for CB_i is considered an inappropriate actuator event and lead to the error state.

b) Controller: One modular controller for each CB_i is designed in an ad-hoc manner. It allows for any sensor event in any state which shall be potentially preempted by actuator events when considering the global behaviour. Each controller is composed from two parts: H_i^a describes the nominal repetition of the sequence $lv_{i-1} \cdot on_i \cdot ar_{i+1} \cdot off_i$. On the other hand, H_i^b utilises lv_i and ar_{i+2} to monitor whether CB_{i+1} is busy; if CB_{i+1} is busy, CB_{i+1} shall halt and wait until CB_{i+1} becomes idle[1]. For the overall controller, the synchronous composition of H_i^a and H_i^b is taken. However, this allows for SCCs with actuator events only, a.k.a. *activity loops*. Since a controller must not have any activity loops, these are manually removed. The resulting controller module for CB_i is denoted H_i.

For each CB_i, the controlled local behaviour amounts to

$$K_i := G_i^a \parallel G_i^b \parallel H_i, \quad 1 \le i \le k, \tag{8}$$

[1]Note that for the right most CB_k, an event ar_{k+2} will be generated which does not belong to any plant component and thus can be neglected through natural projection

with K_0 given by Fig. 5 for the special case of the source. After the first workpiece, the source will only send another workpiece when CB_2 has received the former one. The behaviour of the sink is represented by the automaton G_{k+1}^a. Finally, the global controlled behaviour complies with

$$K := \mathcal{S}(K_0 \parallel G_{k+1}^a \parallel \big\|_{1 \le i \le k} K_i) \tag{9}$$

and we verify non-blockingness by the approach presented in the previous sections. The time elapsed using a PC with an 1.80 GHz Intel i7-10510U CPU and 16 GB RAM is listed in Table II. The first column shows the count of conveyor belts, the second column shows the approximate overall count of the reachable states before shaping. The column "with quotient" shows the time elapsed when applying the complete abstraction procedure as introduced in Section IV. To avoid computation of partitions on generators with exaggerated state count (which may take longer than computing the synchronous composition), partition algorithms are omitted when the input automaton has a state count of more then 10^4. In the column "shape only", we only apply the abstraction "shaping w.r.t. $\{\tau^p\}$". In either case, the employed heuristics for composition consistently chooses the two left most components. Finally, in the last column "no abstraction", we show the elapsed time for verification when no abstraction is applied. As can be seen, without abstraction the elapsed time is considerably longer than with abstraction. In addition, a decent time reduction can be observed in the "shape only" column, since for this example each component automaton only has local preemptive events. Nevertheless, the foremost result is still provided by utilising quotient construction w.r.t. OEn and OEp, which at $k = 10$ reduces the elapsed time to about 20%.

TABLE II

ELAPSED TIME ("–" FOR MORE THAN 10MINUTES)

k	apprx. state cnt.	with quotient	shape only	no abstraction
7	2.0×10^7	1.4s	2.6s	62.5s
8	1.8×10^8	2.7s	7.8s	493.8s
9	1.5×10^9	6.9s	24.3s	–
10	1.3×10^{10}	21.8s	97.8s	–

VI. CONCLUSIONS AND PROSPECTS

In this paper, we have re-visited compositional non-conflictingness verification to account for preemptive events. We have identified quotient-based conflict-preserving abstractions that comply with our notion of preemption and that can be used on a per–component basis. Although not guaranteed in general, computational benefits are expected for relevant applications and have been demonstrated by a practical example. Technically, our contribution is based on adequate refinements of the notion of observation equivalence. In future work, we envisage to suitably adapt further methods of abstraction based on those proposed in [9]. It will also be interesting to discuss how our work relates to the process-algebraic abstractions presented in [21], [22].

References

[1] R. Milner, *Communication and Concurrency*. USA: Prentice-Hall, Inc., 1989.

[2] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dynamic Systems*, vol. 17, 11 2007.

[3] S. Mohajerani, R. Malik, and M. Fabian, "An algorithm for weak synthesis observation equivalence for compositional supervisor synthesis," *IFAC Proceedings Volumes*, vol. 45, no. 29, pp. 239–244, 2012, 11th IFAC Workshop on Discrete Event Systems.

[4] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional synthesis of modular nonblocking supervisors," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 150–162, 2014.

[5] R. Malik, D. Streader, and S. Reeves, "Fair testing revisited: A process-algebraic characterisation of conflicts," in *Automated Technology for Verification and Analysis*, F. Wang, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 120–134.

[6] H. Flordal and R. Malik, "Compositional verification in supervisory control," *SIAM J. Control and Optimization*, vol. 48, pp. 1914–1938, 2009.

[7] R. Su, J. H. van Schuppen, J. E. Rooda, and A. T. Hofkamp, "Nonconflict check by using sequential automaton abstractions based on weak observation equivalence," *Automatica*, vol. 46, no. 6, pp. 968 – 978, 2010.

[8] S. Ware and R. Malik, "Conflict-preserving abstraction of discrete event systems using annotated automata," *Discrete Event Dynamic Systems*, vol. 22, pp. 451–477, 2012.

[9] C. Pilbrow and R. Malik, "An algorithm for compositional nonblocking verification using special events," *Science of Computer Programming*, vol. 113, 06 2015.

[10] R. Malik and R. Leduc, "Compositional nonblocking verification using generalized nonblocking abstractions," *IEEE Transactions on Automatic Control*, vol. 58, no. 8, pp. 1891–1903, 2013.

[11] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional nonblocking verification of extended finite-state machines," *Discrete Event Dynamic Systems*, vol. 9, 09 2015.

[12] S. Mohajerani and S. Lafortune, "Transforming opacity verification to nonblocking verification in modular systems," *IEEE Transactions on Automatic Control*, vol. 65, no. 4, pp. 1739–1746, 2020.

[13] B. Lennartson, X. Liang, and M. Noori Hosseini, "Efficient temporal logic verification by incremental abstraction," 08 2020, pp. 894–899.

[14] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231 – 274, 1987.

[15] C. Brooks and E. Lee, "Ptolemy II: An open-source platform for experimenting with actor-oriented design," *Berkeley EECS Annual Research Symposium (BEARS)*, 2016.

[16] J. Provost, J. Roussel, and J. Faure, "A formal semantics for grafcet specifications," in *2011 IEEE International Conference on Automation Science and Engineering*, 2011, pp. 488–494.

[17] Object Management Group, *OMG Unified Modeling Language*. An OMG Unified Modeling Language Publication, 2017.

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

[19] J. Baeten, J. Bergstra, and J. Klop, *Syntax and defining equations for an interrupt mechanism in process algebra*, ser. CWI report. CS-R. Centrum voor Wiskunde en Informatica, 1985.

[20] "Priorities in process algebras," *Information and Computation*, vol. 87, no. 1, pp. 58–77, 1990, special Issue: Selections from 1988 IEEE Symposium on Logic in Computer Science.

[21] G. Lüttgen, "Pre-emptive modeling of concurrent and distributed systems," 1998.

[22] R. Cleaveland, G. Lüttgen, and V. Natarajan, "Priority and abstraction in process algebra," *Information and Computation*, vol. 205, no. 9, pp. 1426–1458, 2007.

[23] B. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, pp. 329–342, 1994.

[24] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.

[25] A. Boulgakov, T. Gibson-Robinson, and A. Roscoe, "Computing maximal weak and other bisimulations," *Formal Aspects of Computing*, vol. 28, pp. 381–407, 2016.

[26] S. Blom and S. Orzan, "Distributed state space minimization," *Electronic Notes in Theoretical Computer Science*, vol. 80, pp. 109–123, 08 2003.

[27] R. Malik and S. Ware, "Counterexample computation in compositional nonblocking verification," *IFAC-PapersOnLine*, vol. 51, pp. 416–421, 01 2018.

APPENDIX – PROOFS

We prepare our proofs by introducing some additional terminology. A *trace* is a sequence of states related by transitions, e.g.

$$x_0 \xrightarrow{\sigma_1} x_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_k} x_k \tag{10}$$

with length $k+1$. We may also use the terminology *all traces in* $x \overset{s}{\Rightarrow} y$, since x may reach y via the same string by multiple distinct traces. Moreover, for the transitions in the quotient automaton of G w.r.t. $\sim$, we utilise the notation $\rightarrow_{\tilde{G}}$ and $\Rightarrow_{\tilde{G}}$.

A. Proof of Proposition 1 (Consistency)

Let $T = \langle Q_T, \Sigma_T, \rightarrow_T, Q_T^0 \rangle$ be an automaton. ($\Longrightarrow$) Suppose $\mathcal{S}(G \parallel T)$ is non-blocking. Pick any $([x_G], x_T) \in Q_G/\sim \times Q_T$ s.t. $\mathcal{S}(G/\sim \parallel T) \overset{s}{\Rightarrow}^{\mathcal{S}} ([x_G], x_T)$ for some $s \in (\Sigma_G \cup \Sigma_T)^*$. Then by (C1), we observe by induction that there exists some $x'_G \in [x_G]$ s.t. $\mathcal{S}(G \parallel T) \overset{s}{\Rightarrow}^{\mathcal{S}} (x'_G, x_T)$. Since $\mathcal{S}(G \parallel T)$ is non-blocking, it holds $(x'_G, x_T) \overset{t\omega}{\Longrightarrow}^{\mathcal{S}}$ for some $t \in (\Sigma_G \cup \Sigma_T)^*$. Then by induction on (C2), $([x_G], x_T) \overset{t\omega}{\Longrightarrow}^{\mathcal{S}}$ holds. Since $([x_G], x_T)$ is arbitrarily chosen, this establishes that $\mathcal{S}(G/\sim \parallel T)$ is non-blocking. ($\Longleftarrow$) The converse implication follows likewise.

B. Proof of Proposition 2 (OEn)

To obtain Proposition 2, we establish Lemmata 1 and 2 to match (C1) and (C2) of Definition 7, respectively. Note that since $\Rightarrow^n$ implies $\Rightarrow$, it is self-evident that replacing $\Rightarrow$ by $\Rightarrow^n$ (or $\Rightarrow^p$) in Definition 7 yields a strictly stronger version of reachability consistency, which obviously implies conflict preservation under preemption as well. Thus, in this paragraph, since OEn only considers τ^n as silent, we utilize $\Rightarrow$ to denote $\Rightarrow^n$ and p to denote p^n for brevity.

Lemma 1. Let $G = \langle Q_G, \Sigma_G, \rightarrow_G, Q_G^0 \rangle = \mathcal{S}(G)$ be a shaped automaton with an OEn $\sim^n \subseteq Q_G \times Q_G$ on G. For any automaton $T = \langle Q_T, \Sigma_T, \rightarrow_T, Q_T^0 \rangle$, if for any $[x_G], [y_G] \in Q_G/\sim^n$ and $x_T, y_T \in Q_T$, $([x_G], x_T) \overset{\sigma}{\rightarrow}^{\mathcal{S}} ([y_G], y_T)$ for some $\sigma \in \Sigma_G \cup \Sigma_T \cup \{\tau^n, \tau^p, \omega\}$, then for all $x'_G \in [x_G]$, there exists some $y'_G \in [y_G]$ s.t. $(x'_G, x_T) \overset{\mathsf{p}(\sigma)}{\Longrightarrow}^{\mathcal{S}} (y'_G, y_T)$.

Proof. We split the proof into two cases:

(*Case A*) $[x_G] \overset{\sigma}{\rightarrow}_{\tilde{G}} [y_G]$. Note that $\sigma \in \Sigma_G \cup \{\tau^n, \tau^p, \omega\}$. From (N1), we have

$$x'_G \overset{\epsilon}{\Rightarrow}_G \tilde{x}_G \xrightarrow{\mathsf{p}(\sigma)}_G y'_G \tag{11}$$

in G for some $\tilde{x}_G \in Q_G$ and $y'_G \in [y_G]$. This implies that in $G \parallel T$, we have

$$(x'_G, x_T) \overset{\epsilon}{\Rightarrow} (\tilde{x}_G, x_T) \xrightarrow{\mathsf{p}(\sigma)} (y'_G, y_T). \tag{12}$$

If all traces in (12) for all x'_G can be preserved under shaping, we conclude $(x'_G, x_T) \overset{\mathsf{p}(\sigma)}{\Longrightarrow}^{\mathcal{S}} (y'_G, y_T)$. We now prove that (12) will not be affected by shaping, and we do so via two further sub-cases:

(Case A.1) $\sigma \in \Sigma_G^p \cup \{\tau^p\}$. Then $[x_G]$ is a preemptive state and from (N2), x'_G must be preemptive as well. Since $G = \mathcal{S}(G)$, τ^n is not active on x'_G, thus it holds $x'_G = \tilde{x}_G$. Consequently, it follows $(x'_G, x_T) \overset{\sigma}{\rightarrow} (y'_G, y_T)$ in $G \parallel T$. Since this transition is equipped with a preemptive event, it will not be affected by shaping.

(Case A.2) $\sigma \notin \Sigma_G^p \cup \{\tau^p\}$. From (N2), since $G = \mathcal{S}(G)$, it holds that $G/{\sim^n} = \mathcal{S}(G/{\sim^n})$. Thus, $[x_G]$ is not preemptive and x'_G cannot be preemptive either. Since $G = \mathcal{S}(G)$ and σ is not preemptive, all states on any trace in $x'_G \overset{\epsilon}{\Rightarrow}_G \tilde{x}_G$ before $\tilde{x}_G$ are not preemptive. On the other hand, $([x_G], x_T) \overset{\sigma}{\to}^{\mathcal{S}} ([y_G], y_T)$ implies that $([x_G], x_T)$ is not preemptive, which indicates that no private preemptive event or τ^p is active on x_T. Hence, we conclude that all states on any trace in $(x'_G, x_T) \overset{\epsilon}{\Rightarrow} (\tilde{x}_G, x_T)$ before $(\tilde{x}_G, x_T)$ are not preemptive, since even if x_T is preemptive, it can only own active *shared* preemptive events, and they will be steadily deactivated by states before $\tilde{x}_G$ on traces in $x_G \overset{\epsilon}{\Rightarrow}_G \tilde{x}_G$. Furthermore, $(\tilde{x}_G, x_T) \overset{\mathsf{p}(\sigma)}{\longrightarrow} (y'_G, y_T)$ will not be affected by shaping either: if $\sigma = \tau^n$, this will be a trivial transition; otherwise, $\tilde{x}_G \overset{\sigma}{\to}_G y_G$ and, since $G = \mathcal{S}(G)$, $\tilde{x}$ is not preemptive, implying that $(\tilde{x}_G, x_T)$ is not preemptive either.

(*Case B*) $[x_G] \overset{\sigma}{\not\to}_G^{\sim} [y_G]$. Then $x_T \overset{\sigma}{\to}_T y_T$. Note that σ is either a silent event or private in T, which implies

$$(x'_G, x_T) \overset{\sigma}{\to} (x'_G, y_T). \tag{13}$$

Note that $[x_G] = [y_G]$ and thus $x'_G \in [y_G]$. We then show that (13) will not be affected by shaping:

(Case B.1) $\sigma \in \Sigma_T^p \cup \{\tau^p\}$. Then (13) is not affected by shaping.

(Case B.2) $\sigma \notin \Sigma_T^p \cup \{\tau^p\}$. Then $([x_G], x_T)$ is not preemptive, indicating for any $x''_G \in [x_G]$, (x''_G, x_T) is not preemptive, otherwise $([x_G], x_T)$ is preemptive. □

Lemma 2. Let $G = \langle Q_G, \Sigma_G, \to_G, Q_G^0 \rangle = \mathcal{S}(G)$ be a shaped automaton with an OEn $\sim^n \subseteq Q_G \times Q_G$ on G. For any automaton $T = \langle Q_T, \Sigma_T, \to_T, Q_T^0 \rangle$, if for any $x_G, y_G \in Q_G$ and $x_T, y_T \in Q_T$, $(x_G, x_T) \overset{\sigma}{\to}^{\mathcal{S}} (y_G, y_T)$ for some $\sigma \in \Sigma_G \cup \Sigma_T \cup \{\tau^n, \tau^p, \omega\}$, then it holds that $([x_G], x_T) \overset{\sigma}{\to}^{\mathcal{S}} ([y_G], x_T)$.

Proof. From the construction of quotient automata, we obtain $[x_G] \overset{\sigma}{\to}_G^{\sim} [y_G]$. Thus, in $G/{\sim^n} \parallel T$ we have $([x_G], x_T) \overset{\sigma}{\to} ([y_G], y_T)$. We now show that this transition is not affected by shaping.

(*Case A*) σ is preemptive. Then $([x_G], x_T) \overset{\sigma}{\to} ([y_G], y_T)$ is not affected by shaping.

(*Case B*) σ is not preemptive. Then (x_G, x_T) is not preemptive, implying that no private preemptive events or τ^p are active in x_G or x_T. In addition, (x_G, x_T) cannot be synchronised by any shared preemptive event. It now follows that $([x_G], x_T)$ is not preemptive since if so, it must hold $[x_G] \overset{\sigma^p}{\longrightarrow}_G^{\sim}$ in $G/{\sim^n}$ for some $\sigma^p \in \Sigma_G^p \cup \{\tau^p\}$. Since (x_G, x_T) is not preemptive, it follows $x_G \overset{\epsilon}{\Rightarrow}_G \tilde{x}_G \overset{\sigma^p}{\longrightarrow}_G$ where $x_G \neq \tilde{x}_G$. As $G = \mathcal{S}(G)$, we conclude that x_G is not preemptive, which contradicts with (N2) since the case assumption requires $[x_G]$ to be preemptive. Thus, $([x_G], x_T)$ is not preemptive and $([x_G], x_T) \overset{\sigma}{\to}^{\mathcal{S}} ([y_G], y_T)$. □

C. Proof of Proposition 3 (OEp)

We establish Lemmata 3 and 4 to imply Proposition 3. Note that Proposition 3 requires a disjoint set of preemptive events. In this section, $\Rightarrow$ denotes $\Rightarrow^p$ and p denotes p^p since we only focus on abstracting τ^p.

Lemma 3. Let $G = \langle Q_G, \Sigma_G, \to_G, Q_G^0 \rangle = \mathcal{S}(G)$ be a shaped automaton with an OEp $\sim^p \subseteq Q_G \times Q_G$ on G. For any automaton $T = \langle Q_T, \Sigma_T, \to_T, Q_T^0 \rangle$, if for any $[x_G], [y_G] \in Q_G/{\sim^p}$ and $x_T, y_T \in Q_T$, $([x_G], x_T) \overset{\sigma}{\to}^{\mathcal{S}} ([y_G], y_T)$ for some $\sigma \in \Sigma_G \cup \Sigma_T \cup \{\tau^n, \tau^p, \omega\}$, then for all $x'_G \in [x_G]$, there exists some $y'_G \in [y_G]$ s.t. $(x'_G, x_T) \overset{\mathsf{p}(\sigma)}{\Longrightarrow}^{\mathcal{S}} (y'_G, y_T)$.

Proof. We organise the proof by two cases:

(*Case A*) $[x_G] \overset{\sigma}{\to}_G^{\sim} [y_G]$. From (P1), $x'_G \overset{\mathsf{p}(\sigma)}{\Longrightarrow} y'_G$ in G for some $y'_G \in [y_G]$. It follows that in $G \parallel T$ we have $(x'_G, x_T) \overset{\mathsf{p}(\sigma)}{\Longrightarrow} (y'_G, y_T)$. In the following, we show that traces in $(x'_G, x_T) \overset{\mathsf{p}(\sigma)}{\Longrightarrow} (y'_G, y_T)$ will not be affected by shaping. There are two further sub-cases:

(Case A.1) $\sigma \in \Sigma_G^p \cup \{\tau^p\}$, then on any trace in $(x'_G, x_T) \overset{\mathsf{p}^p(\sigma)}{\Longrightarrow}^p (y'_G, y_T)$, all neighbouring states are via transitions with preemptive events.

(Case A.2) $\sigma \notin \Sigma_G^p \cup \{\tau^p\}$, then $([x_G], x_T)$ is not preemptive. This implies $x''_G \overset{\tau^p}{\not\to}_G$ for any $x''_G \in [x_G]$. Thus, $x'_G \overset{\sigma}{\to}_G$ and $(x'_G, x_T) \overset{\sigma}{\to}$. It also follows that (x'_G, x_T) cannot be preemptive, since if so, $([x_G], x_T)$ would be preemptive, too. To this stage, we have in $G \parallel T$ either directly $(x'_G, x_T) \overset{\sigma}{\to} (y'_G, y_T)$ or $(x'_G, x_T) \overset{\sigma}{\to} (\tilde{x}_G, y_T) \overset{\epsilon}{\Rightarrow} (y'_G, y_T)$ for some $\tilde{x}_G \in Q_G$. In the latter case, all neighbouring states in any trace of $(\tilde{x}_G, y_T) \overset{\epsilon}{\Rightarrow} (y'_G, y_T)$ are connected with τ^p transitions, which obviously will not be affected by shaping.

(*Case B*) $[x_G] \overset{\sigma}{\not\to}_G^{\sim} [y_G]$ in $G/{\sim}$. The proof of this case is identical to that of Case B in Lemma 1. □

Lemma 4. Let $G = \langle Q_G, \Sigma_G, \to_G, Q_G^0 \rangle$ be an automaton with an OEp $\sim^p \subseteq Q_G \times Q_G$ on G. For any automaton $T = \langle Q_T, \Sigma_T, \to_T, Q_T^0 \rangle$ s.t. $\Sigma_G^p \cap \Sigma_T^p = \emptyset$, if for any $x_G, y_G \in Q_G$ and $x_T, y_T \in Q_T$, $(x_G, x_T) \overset{\sigma}{\to}^{\mathcal{S}} (y_G, y_T)$ for some $\sigma \in \Sigma_G \cup \Sigma_T \cup \{\tau^n, \tau^p, \omega\}$, it holds $([x_G], x_T) \overset{\sigma}{\to}^{\mathcal{S}} ([y_G], x_T)$.

Proof. From the construction of quotient automata, it holds $[x_G] \overset{\sigma}{\to}_G^{\sim} [y_G]$. Thus, in $G/{\sim^n} \parallel T$, we have $([x_G], x_T) \overset{\sigma}{\to} ([y_G], y_T)$. We then show that this transition will not be eliminated by shaping.

(*Case A*) σ is preemptive. Then $([x_G], x_T) \overset{\sigma}{\to} ([y_G], y_T)$ will trivially not be eliminated by shaping.

(*Case B*) σ is not preemptive. Then (x_G, x_T) is not preemptive. Since $\Sigma_G^p \cap \Sigma_T^p = \emptyset$, it follows that neither x_G nor x_T is preemptive. From (P2), it follows that $[x_G]$ is not preemptive, thus $([x_G], x_T)$ is not preemptive either. □

D. Proof of Proposition 4 (SOEp)

We establish Lemma 5 to imply Proposition 4. Note that Lemma 5 only matches (C2) of reachability consistency. Nevertheless, since SOEp is finer then OEp, a proposition similar to Proposition 3 will trivially hold, i.e. (C1) is trivially true for SOEp on any automaton. In this paragraph, $\Rightarrow$ denotes $\Rightarrow^p$ and p denotes p^p since we only focus on abstracting τ^p.

Lemma 5. Let $G = \langle Q_G, \Sigma_G, \to_G, Q_G^0 \rangle$ be an automaton with an SOEp $\approx^p \subseteq Q_G \times Q_G$ on G. For any automaton $T = \langle Q_T, \Sigma_T, \to_T, Q_T^0 \rangle$, if for any $x_G, y_G \in Q_G$ and $x_T, y_T \in Q_T$, $(x_G, x_T) \overset{\sigma}{\to}^{\mathcal{S}} (y_G, y_T)$ for some $\sigma \in \Sigma_G \cup \Sigma_T \cup \{\tau^n, \tau^p, \omega\}$, it holds $([x_G], x_T) \overset{\sigma}{\to}^{\mathcal{S}} ([y_G], x_T)$.

Proof. We review Case B in the proof for Lemma 4, since this is the only case which refers to the restriction $\Sigma_G \cap \Sigma_T = \emptyset$. Given a non-preemptive event σ, the state (x_G, x_T) clearly is non-preemptive, too. This implies $x_G \overset{\tau^p}{\not\to}$. From (S2), it follows $([x_G], x_T) \overset{\tau^p}{\not\to}$ and, consequently, $([x_G], x_T)$ can not be preemptive since all $x''_G \in [x_G]$ must have the same set of active events. □