Noname manuscript No.
(will be inserted by the editor)

On the Computation of Counterexamples in Compositional Nonblocking Verification

Robi Malik · Simon Ware

the date of receipt and acceptance should be inserted later

Abstract This paper describes algorithms to compute a counterexample when compositional nonblocking verification determines that a discrete event system is blocking. Counterexamples are an important feature of model checking that explains the cause of a detected problem, greatly helping users to understand and fix faults. In compositional verification, counterexamples are difficult to compute due to the large state space and the loss of information after abstraction. The paper explains the difficulties and proposes solutions, and experimental results show that counterexamples can be computed successfully for several industrial-scale systems.

Keywords Model checking, Compositional verification, Discrete event systems, Nonblocking.

1 Introduction

The *nonblocking property* is a weak liveness property commonly used in *supervisory control theory* of discrete event systems to express the absence of livelocks and deadlocks [22]. This is a crucial property of safety-critical control systems, and with the increasing size and complexity of these systems, there is an increasing need to verify them automatically. The standard method to check the nonblocking property involves the explicit composition of all components involved, and is limited by the well-known *state-space explosion* problem. *Symbolic model checking* can be used to reduce the memory requirements by representing the state space symbolically rather than enumerating it explicitly [2].

Compositional verification [4, 8, 24] is an effective alternative that can be used independently of or in combination with symbolic methods. Compositional verification exploits the fact that large systems are typically modelled by several components interacting in synchronous composition. Then *compositional minimisation*

Robi Malik
Department of Computer Science, University of Waikato, Hamilton, New Zealand
E-mail: robi@waikato.ac.nz

Simon Ware
E-mail: simianware@gmail.com

or *abstraction* [4] can be used to simplify individual components before computing their synchronous composition, gradually reducing the state space of the system and allowing much larger systems to be verified in the end. The ways how components can be simplified to ensure correct verification results depends on the property being verified [5].

The nonblocking property considered in this paper is logically different from most properties commonly studied for compositional verification, and requires specific abstraction methods [7]. A suitable theory is laid out in previous work [17], where it is argued that abstractions used in nonblocking verification should preserve a process-algebraic equivalence called *conflict equivalence*.

Various abstraction rules preserving conflict equivalence have been proposed and used for compositional nonblocking verification. First, *observer projection* [20], *weak observation equivalence* [23], and the *set of certain conflicts* [13, 14] have been used to simplify individual components. The journal paper [7] introduces conflict equivalence to compositional nonblocking verification and proposes a more comprehensive set of conflict-preserving abstraction rules. The same technique has also been applied to compositional verification of the *generalised nonblocking* property [16], giving rise to an improved set of abstraction rules. It has also been proposed to replace abstraction rules by more general simplification processes using *annotated automata* [26] or *canonical automata* [27]. All these methods are based on conflict equivalence, and make no assumptions about the system components not being simplified. To improve the degree of simplification, it has also been proposed to use a weaker equivalence that takes more information about the remainder of the system into account, e.g., by identifying events that are used in special ways [21].

The above publications on compositional nonblocking verification describe algorithms that decide efficiently for a given system whether or not it is nonblocking. If a system fails this nonblocking check, it is also important to present a *counterexample* that shows the cause of the problem and helps to find a fix. The counterexample is a sequence of events that takes the system to a livelock or deadlock situation, and it is routinely computed by standard model checking algorithms [15]. However, if verification is done compositionally, the counterexample at first only applies to the simplified system, and needs to be converted back to the original system. This problem has been addressed for safety properties [25], but only partly [13] for the nonblocking property.

This paper gives a more complete account of the issues and solutions regarding counterexample computation in compositional nonblocking verification. It is an extended version of its conference precursor [18]. It includes the new Section 3.4 that shows how some particular abstraction rules fit in the framework of the paper, and the new Section 5 with a general counterexample algorithm. Also included are full formal proofs of all technical results.

In the following, Section 2 reviews the background of finite-state machines, the nonblocking property, and compositional verification. Then Section 3 describes the process of counterexample computation for two common classes of abstraction rules, and Section 4 shows experimental results for these methods. Afterwards, Section 5 proposes a general algorithm independent of specific abstraction rules that works for all conflict-preserving abstractions. Lastly, Section 6 adds concluding remarks.

2 Preliminaries

2.1 Languages and Finite-State Machines

Event sequences and languages are a simple means to describe discrete system behaviours [3, 22]. Their basic building blocks are *events*, which are taken from a finite *alphabet* Σ. Two special events are used, the *silent event* τ and the *termination event* ω. These are never included in an alphabet Σ unless mentioned explicitly using notation such as $\Sigma_\tau = \Sigma \cup \{\tau\}$, $\Sigma_\omega = \Sigma \cup \{\omega\}$, and $\Sigma_{\tau,\omega} = \Sigma \cup \{\tau, \omega\}$.

$\Sigma^*_{\tau,\omega}$ denotes the set of all finite *traces* of the form $\sigma_1 \cdots \sigma_n$ of events from $\Sigma_{\tau,\omega}$, including the *empty trace* ε. A subset $L \subseteq \Sigma^*_{\tau,\omega}$ is called a *language*. The *concatenation* of two traces $s, t \in \Sigma^*_{\tau,\omega}$ is written as st. A trace $s \in \Sigma^*_{\tau,\omega}$ is called a *prefix* of $t \in \Sigma^*_{\tau,\omega}$, written $s \sqsubseteq t$, if there exists $u \in \Sigma^*_{\tau,\omega}$ such that $su = t$. The *length* of a trace $s \in \Sigma^*_{\tau,\omega}$, i.e., its number of events, is denoted by $|s|$. The *natural projection* $P\colon \Sigma^*_{\tau,\omega} \to \Sigma^*_\omega$ is the operation that deletes all silent (τ) events from traces.

System behaviours are modelled using finite-state machines. Typically, system models are deterministic, but abstraction may result in nondeterminism.

Definition 1 A (nondeterministic) *finite-state machine (FSM)* is a tuple $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ where Σ is a set of *events*, Q is a finite set of *states*, $\to \subseteq Q \times \Sigma_{\tau,\omega} \times Q$ is the *transition relation*, and $Q^\circ \subseteq Q$ is the set of *initial states*.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$ and extended to traces $s \in \Sigma^*_{\tau,\omega}$ in the standard way. For state sets $X, Y \subseteq Q$, the notation $X \xrightarrow{s} Y$ means $x \xrightarrow{s} y$ for some $x \in X$ and $y \in Y$. For states or state sets x and y, the notation $x \to y$ means $x \xrightarrow{s} y$ for some $s \in \Sigma^*_{\tau,\omega}$, and $x \xrightarrow{s}$ means $x \xrightarrow{s} z$ for some $z \in Q$. Events not in the event set of an FSM are always enabled without state change, so the transition relation is further extended by $x \xrightarrow{\sigma} x$ for all $x \in Q$ and $\sigma \notin \Sigma_{\tau,\omega}$.

To support silent events, another transition relation $\Rightarrow \subseteq Q \times \Sigma^*_\omega \times Q$ is introduced, where $x \stackrel{s}{\Rightarrow} y$ denotes the existence of a trace $t \in \Sigma^*_{\tau,\omega}$ such that $P(t) = s$ and $x \xrightarrow{t} y$. That is, $x \xrightarrow{s} y$ denotes a path with *exactly* the events in s, while $x \stackrel{s}{\Rightarrow} y$ denotes a path with an arbitrary number of τ events shuffled with the events of s. Notations such as $X \stackrel{s}{\Rightarrow} Y$ and $x \stackrel{s}{\Rightarrow}$ are defined analogously to $\to$. For an FSM $G = \langle \Sigma, Q, \to, Q^\circ \rangle$, the notation $G \stackrel{s}{\Rightarrow} x$ means $Q^\circ \stackrel{s}{\Rightarrow} x$.

The termination event $\omega \notin \Sigma$ denotes completion of a task and does not appear anywhere else but to mark such completions. It is required that states reached by ω do not have any outgoing transitions, i. e., if $x \xrightarrow{\omega} y$ then there does not exist $\sigma \in \Sigma_{\tau,\omega}$ such that $y \xrightarrow{\sigma}$. This ensures that the termination event, if it occurs, is always the final event of any trace. The traditional set of *accepting* states is $Q^\omega = \{\, x \in Q \mid x \xrightarrow{\omega} \,\}$ in this notation. The *marked* or *accepting language* of a state or state set x is $\mathcal{L}^\omega(x) = \{\, s \in \Sigma^* \mid x \stackrel{s\omega}{\Rightarrow} \,\}$, and the accepting language of an FSM G is $\mathcal{L}^\omega(G) = \mathcal{L}^\omega(Q^\circ)$. The accepting language contains traces that can be extended to termination with an ω-event, but it does not explicitly include ω.

Example 1 Fig. 1 shows the same FSM G in two graphical representations. The diagram on the left includes ω-transitions that lead to a terminated state, also called ω, without any outgoing transitions. The diagram on the right uses a more conventional representation, where accepting states in Q^ω are coloured black without explicitly showing ω-transitions. The second representation will be used in later figures for graphical simplicity.

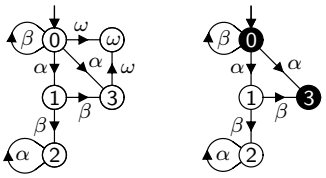

Fig. 1 Graphical representation of an FSM with and without ω-transitions.

FSMs are synchronised in lock-step [10]. The *synchronous composition* of two FSMs $G = \langle \Sigma_G, Q_G, \rightarrow_G, Q_G^\circ \rangle$ and $H = \langle \Sigma_H, Q_H, \rightarrow_H, Q_H^\circ \rangle$ is

$$G \parallel H \;=\; \langle \Sigma_G \cup \Sigma_H, Q_G \times Q_H, \rightarrow, Q_G^\circ \times Q_H^\circ \rangle \tag{1}$$

where

$$(x_G, x_H) \xrightarrow{\sigma} (y_G, y_H) \text{ if } \sigma \neq \tau,\ x_G \xrightarrow{\sigma}_G y_G,\ x_H \xrightarrow{\sigma}_H y_H\ ; \tag{2}$$

$$(x_G, x_H) \xrightarrow{\tau} (y_G, x_H) \text{ if } x_G \xrightarrow{\tau}_G y_G\ ; \tag{3}$$

$$(x_G, x_H) \xrightarrow{\tau} (x_G, y_H) \text{ if } x_H \xrightarrow{\tau}_H y_H\ . \tag{4}$$

Equation (2) uses the extended definition of the transition relation, $x \xrightarrow{\sigma} x$ for $\sigma \notin \Sigma_{\tau,\omega}$, to define synchronous composition for events that appear in only one of the FSMs G or H. As a result, shared events in Σ_ω (including ω) must be executed by both the composed FSMs together, while events that appear in only one FSM and τ are executed by only one FSM without the other changing its state.

2.2 The Nonblocking Property

The key liveness property in supervisory control theory is the *nonblocking* property [22]. An FSM is nonblocking if termination is possible from every reachable state.

Definition 2 [17] An FSM $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ is *nonblocking* if, for every state $x \in Q$ and every trace $s \in \Sigma^*$ such that $Q^\circ \stackrel{s}{\Rightarrow} x$, there exists a trace $t \in \Sigma^*$ such that $x \stackrel{t\omega}{\Rightarrow}$; otherwise G is *blocking*.

This definition generalises the language-based definition [22] of the nonblocking property to the case of nondeterministic state machines. According to the language-based definition, a deterministic FSM is nonblocking if every trace of its behaviour can be extended to an accepting trace, or equivalently for every reachable state there exists a path to a marked or accepting state. This becomes equivalent to Def. 2 if the set of accepting states is considered as the set of states where the termination event ω is enabled, $Q^\omega = \{\, x \in Q \mid x \xrightarrow{\omega} \,\}$. Then Def. 2 means that for every reachable state x it holds that $x \rightarrow Q^\omega$, i.e., an accepting state can be reached from x. (There is an exception for states reached after termination, which do not exist in the language-based setting and which given $\omega \notin \Sigma$ are ruled out by the requirement $Q^\circ \stackrel{s}{\Rightarrow} x$ with $s \in \Sigma^*$.)

If a system is found to be blocking by automatic verification, it is desirable to present an explanation of the fault to the designers. In model checking, this explanation is provided in the form of a *counterexample*. For deterministic FSMs,

counterexamples can be described as traces of events, while in the nondeterministic case the state information is also important. Therefore, this paper considers counterexamples to be *paths*.

Definition 3 Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an FSM. A *path* in G is an alternating sequence of states and events of G,

$$C : x_0 \stackrel{\sigma_1}{\to} x_1 \stackrel{\sigma_2}{\to} \cdots \stackrel{\sigma_n}{\to} x_n \ . \tag{5}$$

The path C is said to be *accepted* by G if $x_0 \in Q^\circ$ and $x_{i-1} \stackrel{\sigma_i}{\to} x_i$ for $i = 1, \ldots, n$. The *event trace* of the path C is the sequence of its events, $\mathrm{trace}(C) = \sigma_1 \cdots \sigma_n$, and the *length* of the path C is its number of events, $|C| = |\mathrm{trace}(C)| = n$.

Definition 4 Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an FSM. A *counterexample to the nonblocking property* of G is a path $x_0 \stackrel{\sigma_1}{\to} \cdots \stackrel{\sigma_n}{\to} x_n$ accepted by G such that $\mathcal{L}^\omega(x_n) = \emptyset$.

A counterexample highlights the cause of blocking by showing a path that leads to a faulty (blocking) state. It starts from the initial state and follows transitions of the FSM. Its end state x_n has an empty accepting language, $\mathcal{L}^\omega(x_n) = \emptyset$, or equivalently $x_n \stackrel{t\omega}{\Rightarrow}$ does not hold for any $t \in \Sigma^*$. Such a state is called a *blocking state*. Clearly, a counterexample to the nonblocking property exists if and only if G is blocking.

Example 2 The FSM in Fig. 1 is blocking, because the reachable state 2 is blocking. Although the system can still execute the transition $2 \stackrel{\alpha}{\to} 2$, it will never be able to terminate from state 2 so that $\mathcal{L}^\omega(2) = \emptyset$. A counterexample to the nonblocking property is

$$0 \stackrel{\alpha}{\to} 1 \stackrel{\beta}{\to} 2 \ . \tag{6}$$

2.3 The Conflict Preorder

To reason about the nonblocking property in a compositional way, the notion of *conflict equivalence* is used [17]. According to process-algebraic testing theory, two FSMs are considered as equivalent if they both respond in the same way to tests [6]. For *conflict equivalence*, a *test* is an arbitrary FSM, and the *response* is the observation whether the test composed with the FSM in question is nonblocking or not.

Definition 5 [17] Let G and H be two FSMs. H is *less conflicting* than G, written $H \lesssim_{\mathrm{conf}} G$, if for any FSM T such that $G \parallel T$ is nonblocking, it also holds that $H \parallel T$ is nonblocking. G and H are *conflict equivalent*, $G \simeq_{\mathrm{conf}} H$, if $G \lesssim_{\mathrm{conf}} H$ and $H \lesssim_{\mathrm{conf}} G$.

If $H \lesssim_{\mathrm{conf}} G$, then H is said to be less conflicting than G, or G is *more conflicting* than H. The properties of the conflict preorder $\lesssim_{\mathrm{conf}}$ and of conflict equivalence and their relationship to other process-algebraic relations are studied in [17]. Conflict equivalence is the coarsest congruence with respect to synchronous composition that respects blocking, making it an ideal equivalence for use in compositional verification of this property [7].

As a related concept, every FSM can be associated with a language of *certain conflicts*, which also plays an important role in conflict semantics [14].

Definition 6 [14] For an FSM $G = \langle \Sigma, Q, \to, Q^\circ \rangle$, write

$$\textsc{Conf}(G) = \{\, s \in \Sigma^* \mid \text{for every FSM } T \text{ such that } T \stackrel{s}{\Rightarrow}, \text{ it holds that } G \parallel T \text{ is blocking} \,\} , \tag{7}$$

$$\textsc{NConf}(G) = \{\, s \in \Sigma^* \mid \text{there exists an FSM } T \text{ such that } T \stackrel{s}{\Rightarrow} \text{ and } G \parallel T \text{ is nonblocking} \,\} . \tag{8}$$

$\textsc{Conf}(G)$ is the set of *certain conflicts* of G. It contains all traces that, when possible in the environment, necessarily cause blocking. Its complement $\textsc{NConf}(G)$ is the most general behaviour of FSMs that are to be nonconflicting with G. If G is nonblocking, then $\textsc{Conf}(G) = \emptyset$ and $\textsc{NConf}(G) = \Sigma^*$, because in this case $G \parallel U$ is nonblocking, where U is a deterministic FSM such that $\mathcal{L}^\omega(U) = \Sigma^*$. The set of certain conflicts becomes interesting for blocking FSMs.

When verifying whether a composed system of FSMs

$$G_1 \parallel G_2 \parallel \cdots \parallel G_n , \tag{9}$$

is nonblocking, *compositional* methods [7, 8] avoid building the full synchronous composition. First, individual FSMs G_i are simplified and replaced by smaller conflict equivalent FSMs. When no further simplification is possible, a subsystem $(G_j)_{j \in J}$ is selected and replaced by its synchronous composition. The result is then simplified again before proceeding further.

The final result of this process is a single FSM H, which is the compositional abstraction of (9). The *congruence* properties [17] of conflict equivalence ensure that H is nonblocking if and only if the original system (9) is. H typically has fewer states than (9), making it possible to check whether it is nonblocking even though the full composition (9) may be too large to fit in memory.

3 Counterexample Expansion

Assume that a system (9) is found to be blocking at the end of compositional verification, i.e., the final compositional abstraction is blocking. Then standard state exploration algorithms [15] produce a counterexample to the nonblocking property in addition to detecting blocking, but this counterexample applies to the compositional abstraction only. After several steps of simplification, it is not guaranteed to apply to the original system (9).

In the following, a counterexample to the nonblocking property of an abstracted system is called an *abstract* counterexample, while a counterexample to the nonblocking property of the system before abstraction is called a *concrete* counterexample.

The fact that each abstraction step preserves conflict equivalence guarantees that, for each abstract counterexample there must exist a concrete counterexample. A concrete counterexample for the original system (9) can be obtained by a process of *expansion*. Starting with the last abstraction step, the abstract counterexample is modified to be a concrete counterexample for the system before the last step, and this is repeated for each abstraction step until the original system is reached. Precisely how these expansion steps work depends on the particular kind of abstraction performed at each step.

Next, Section 3.1 explains the principles of counterexample computation using simple abstraction steps. Then Section 3.2 presents the counterexample expansion algorithm, Section 3.3 proves its correctness, and Section 3.4 discusses the conflict-preserving abstraction rules that work together with this algorithm. Afterwards, Section 3.5 describes the more complicated counterexample extension algorithm, which is needed for some abstraction rules.

3.1 Synchronous Composition and Hiding

The simplest abstraction step is that of *synchronous composition*. The system (9) can be replaced by

$$(G_1 \parallel G_2) \parallel G_3 \parallel \cdots \parallel G_n \ . \tag{10}$$

Here, two components G_1 and G_2 are selected and replaced by their synchronous composition. Clearly, the composed systems before and after abstraction are isomorphic in this case, so any abstract counterexample is also a concrete counterexample—except for the state information. The state tuples in an abstract counterexample for (10) have the structure $((x_1, x_2), x_3, \ldots, x_n)$, which needs to be changed to $(x_1, x_2, x_3, \ldots, x_n)$ in a concrete counterexample for (9).

Another simple type of abstraction is *hiding*. Assume that in (9) the events in some subset $\Upsilon \subseteq \Sigma$ appear only in G_1 and in no other components. Then (9) can be replaced by

$$(G_1 \setminus \Upsilon) \parallel G_2 \parallel \cdots \parallel G_n \ . \tag{11}$$

Here, $G_1 \setminus \Upsilon$ is the result of hiding, which is obtained from G_1 by replacing all events in Υ by the silent event τ [7]. The abstraction is isomorphic to the original system apart from event renaming. An abstract counterexample for (11) may contain steps labelled τ that are not possible in the original system (9). These steps can be identified from the state information, and their τ events must be replaced with the correct events from the original FSM G_1.

These two counterexample transformations are straightforward, provided that there is sufficient information about the intermediate FSMs computed during compositional abstraction. This information must either be held in memory or recalculated on demand.

3.2 State Merging

A common method to simplify an FSM is to construct its *quotient* modulo an equivalence relation. The following definitions are standard.

An *equivalence relation* is a reflexive, symmetric, and transitive relation. Given an equivalence relation $\sim$ on a set Q, the *equivalence class* of $x \in Q$ with respect to $\sim$ is $[x] = \{\, x' \in Q \mid x' \sim x \,\}$. An equivalence relation on a set Q partitions Q into $Q/{\sim} = \{\, [x] \mid x \in Q \,\}$.

Definition 7 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim \ \subseteq Q \times Q$ be an equivalence relation. The *quotient FSM* $G/{\sim}$ of G with respect to $\sim$ is $G/{\sim} = \langle \Sigma, Q/{\sim}, \rightarrow/{\sim}, \tilde{Q}^\circ \rangle$, where $\rightarrow/{\sim} = \{\, ([x], \sigma, [y]) \mid x \xrightarrow{\sigma} y \,\}$ and $\tilde{Q}^\circ = \{\, [x^\circ] \mid x^\circ \in Q^\circ \,\}$.

When constructing a quotient FSM, classes of equivalent states are combined or *merged* into a single state. The quotient FSM contains a transition linking two classes of states if the original FSM contains a transition with the same event that links some states of these classes.

There are several relations $\sim$ such that G_1 and $G_1/\!\sim$ are conflict equivalent [7,23]. Therefore, it is common in compositional nonblocking verification to replace an FSM in (9) by its quotient and obtain an abstract system

$$(G_1/\!\sim) \parallel G_2 \parallel \cdots \parallel G_n \ . \tag{12}$$

If the abstracted system (12) is blocking, then it has a counterexample accepted by all its components that ends in a blocking state. This counterexample needs to be modified so that it is accepted by G_1 rather than $G_1/\!\sim$ and leads to an end state that is blocking in the original system (9). The following condition ensures that this counterexample modification can be done using information about G_1 and $G_1/\!\sim$ only.

Definition 8 Let G and H be two FSMs. H is *counterexample-based less conflicting* than G, written $H \lesssim_{\text{ce}} G$, if for all paths $H \stackrel{s}{\Rightarrow} x_H$ with $s \in \Sigma^*$ there exists a state x_G of G such that $G \stackrel{s}{\Rightarrow} x_G$ and $\mathcal{L}^\omega(x_G) \subseteq \mathcal{L}^\omega(x_H)$.

The abstraction H is counterexample-based less conflicting than the original FSM G, if for every state reachable in the abstraction, the original FSM has a state reached by the same event trace, such that the marked language of the state in the original FSM is contained in the marked language of the state of the abstraction. This state-based property can be shown to be stronger than the conflict preorder.

Proposition 1 Let G and H be two FSMs. If $H \lesssim_{\text{ce}} G$ then $H \lesssim_{\text{conf}} G$.

Proof Let $H \lesssim_{\text{ce}} G$. To show $H \lesssim_{\text{conf}} G$, let T be an FSM such that $G \parallel T$ is nonblocking. It is to be shown that $H \| T$ is nonblocking, so assume $H \| T \stackrel{s}{\Rightarrow} (x_H, x_T)$ for some $s \in \Sigma^*$. Then $H \stackrel{s}{\Rightarrow} x_H$, so by Def. 8 there exists a state x_G of G such that $G \stackrel{s}{\Rightarrow} x_G$ and $\mathcal{L}^\omega(x_G) \subseteq \mathcal{L}^\omega(x_H)$. It follows that $G \parallel T \stackrel{s}{\Rightarrow} (x_G, x_T)$, and since $G \parallel T$ was assumed nonblocking, there exists $t \in \mathcal{L}^\omega((x_G, x_T)) = \mathcal{L}^\omega(x_G) \cap \mathcal{L}^\omega(x_T) \subseteq \mathcal{L}^\omega(x_H) \cap \mathcal{L}^\omega(x_T) = \mathcal{L}^\omega((x_H, x_T))$. As $H \parallel T \stackrel{s}{\Rightarrow} (x_H, x_T)$ was chosen arbitrarily, it follows that $H \parallel T$ is nonblocking. □

If Def. 8 holds, a concrete counterexample can be obtained by the following observations: if an abstract counterexample takes the abstract FSM H to some state x_H, i.e., $H \stackrel{s}{\Rightarrow} x_H$, then the definition ensures the existence of a state x_G of the concrete FSM G with a smaller marked language. This state x_G is a suitable end state for a concrete counterexample. It remains to change the path to x_H so that it ends in x_G while using the same non-τ events, which must be possible because $G \stackrel{s}{\Rightarrow} x_G$.

Algorithm 1 is a search procedure to find a concrete counterexample when Def. 8 is satisfied after abstraction of G_1 in a composition (9). First, the loop on lines 1–3 deletes from the abstract counterexample $\tilde{C}$ all τ-transitions that correspond to the abstract FSM, as these must be replaced by transitions from the concrete FSM. Then line 4 decomposes the resulting path $\tilde{C}$, in particular its length is k. From line 5, the algorithm performs a search through the concrete FSM G_1

Algorithm 1: Counterexample Expansion after State Merging

Input: $G_1, \ldots, G_n$ where $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_i^\circ \rangle$, and $\tilde{G}_1 = G_1/\!\sim$
Input: abstract counterexample $\tilde{C}$ for $\tilde{G}_1 \parallel G_2 \parallel \cdots \parallel G_n$
Output: concrete counterexample C for $G_1 \parallel \cdots \parallel G_n$

while $\tilde{C}$ includes a transition $(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n) \xrightarrow{\tau} (\tilde{z}_1, \tilde{x}_2, \ldots, \tilde{x}_n)$ **do**
  delete the first such transition together with its source state from $\tilde{C}$
end
let $\tilde{C} : \tilde{y}^0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} \tilde{y}^k$, with $\tilde{y}^i = (\tilde{x}_1^i, \tilde{x}_2^i, \ldots, \tilde{x}_n^i)$ for $i = 0, \ldots, k$
foreach $x_1^\circ \in Q_1^\circ$ **do**
  $C := (x_1^\circ, \tilde{x}_2^0, \ldots, \tilde{x}_n^0)$ *// one-state path*
  add $(C, 0)$ to *Queue*
  add $(x_1^\circ, 0)$ to *Visited*
end
while *Queue* is not empty **do**
  remove (C, c) with $|C| + k - c$ minimal from *Queue*
  let $C : y^0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} y^m$, with $y^i = (x_1^i, x_2^i, \ldots, x_n^i)$ for $i = 0, \ldots, m$
  if $c = k$ **then**
    if $\mathcal{L}^\omega(x_1^m) \subseteq \mathcal{L}^\omega(\tilde{x}_1^k)$ **then**
      return C
    end
  else if $\sigma_{c+1} \in \Sigma_1$ **then**
    foreach transition $x_1^m \xrightarrow{\sigma_{c+1}}_1 z_1$ in G_1 **do**
      if $(z_1, c+1) \notin$ *Visited* **then**
        $C' := C \xrightarrow{\sigma_{c+1}} (z_1, \tilde{x}_2^{c+1}, \ldots, \tilde{x}_n^{c+1})$
        add $(C', c+1)$ to *Queue*
        add $(z_1, c+1)$ to *Visited*
      end
    end
  else if $(x_1^m, c+1) \notin$ *Visited* **then**
    $C' := C \xrightarrow{\sigma_{c+1}} (x_1^m, \tilde{x}_2^{c+1}, \ldots, \tilde{x}_n^{c+1})$
    add $(C', c+1)$ to *Queue*
    add $(x_1^m, c+1)$ to *Visited*
  end
  foreach transition $x_1^m \xrightarrow{\tau}_1 z_1$ in G_1 **do**
    if $(z_1, c) \notin$ *Visited* **then**
      $C' := C \xrightarrow{\tau} (z_1, \tilde{x}_2^c, \ldots, \tilde{x}_n^c)$
      add (C', c) to *Queue*
      add (z_1, c) to *Visited*
    end
  end
end

to find paths using the same steps as the abstract counterexample, possibly interleaved with τ-transitions of the concrete FSM. It uses a *Queue* of pairs (C, c), where C is a partially constructed initial segment of a concrete counterexample, and c is the number of events of the abstract counterexample $\tilde{C}$ already processed. The set *Visited* contains pairs (x_1, c) of concrete states x_1 of G_1 and numbers c of processed events, to ensure termination by avoiding duplicate search states. The loop on lines 5–9 initialises the search with concrete counterexamples starting from each initial state of G_1. The main loop on lines 10–37 retrieves each pair (C, c), decomposes C as per line 12, so its length is m, and explores transitions originating from the concrete end state x_1^m in G_1.

If all events of $\tilde{C}$ have been processed, then $c = k$ and line 14 checks the termination condition $\mathcal{L}^\omega(x_1^m) \subseteq \mathcal{L}^\omega(\tilde{x}_1^k)$ according to Def. 8, and returns C as the concrete counterexample if satisfied. Otherwise, the next event σ_{c+1} from $\tilde{C}$ is considered, and its possible transitions are explored to extend C and form new search states. The event is either synchronised with G_1 (lines 17–24) or only performed by the rest of the system $G_2 \| \cdots \| G_n$ (lines 25–28). In any case, τ-transitions in G_1 also have to be explored (lines 30–36).

Assuming Def. 8, Algorithm 1 always terminates through line 15 before the *Queue* becomes empty. The measure $|C| + k - c$ on line 11 optimistically estimates counterexample length, given by the number $|C|$ of events in the constructed concrete counterexample plus the number $k - c$ of events from the abstract counterexample still to be added. By processing pairs where this measure is minimal first, the algorithm performs an A^**-search* that guarantees a shortest result [9].

The language containment check on line 14 is a potentially time-consuming operation. The worst-case time to determine whether $\mathcal{L}^\omega(G) \subseteq \mathcal{L}^\omega(H)$ for nondeterministic FSMs G and H is exponential in the number of states of H. Fortunately, most abstractions satisfy Def. 8 in such a way that this check can be replaced by a simple constant-time operation, as will be shown in Subsection 3.4 below.

Setting aside the time taken for the language containment checks on line 14, the time complexity of Algorithm 1 depends on the length of the abstract counterexample and the size of the concrete FSM. The number of possible search states and thus iterations of the main loop starting on line 10 is bounded by $|\tilde{C}||Q_1|$, and each iteration may process up to two transitions to each state of the nondeterministic FSM G_1 through the loops starting on lines 18 and 30. This gives a worst-case time complexity of $O(|\tilde{C}||Q_1|^2)$, which is insignificant compared to the overall runtime of compositional nonblocking verification.

3.3 Correctness Proof of Counterexample Expansion

This section gives a formal proof of the correctness of Algorithm 1. This algorithm performs a search with a complex loop invariant, which is proved in three steps. First, Lemma 2 establishes the *partial correctness*, which means that the algorithm produces a correct result if it terminates [12]. Afterwards, Lemmas 3 and 4 prove termination. These results are combined in Prop. 5, which states that Algorithm 1 is *totally correct*, i.e., it terminates and gives a correct result.

First, Lemma 2 establishes the partial correctness of Algorithm 1. It shows that any result returned from line 15 is a concrete counterexample. Interestingly, this lemma does not require the assumption of the counterexample-based preorder,

$\tilde{G}_1 \lesssim_{\mathrm{ce}} G_1$. Any counterexample returned by Algorithm 1 is correct, however the algorithm is not guaranteed to return a result unless $\tilde{G}_1 \lesssim_{\mathrm{ce}} G_1$.

Lemma 2 Assume that Algorithm 1 is run with a counterexample to the nonblocking property of $\tilde{G}_1 \parallel G_2 \parallel \cdots \parallel G_n$ as input. If the algorithm returns a counterexample C from line 15, then C is a concrete counterexample to the nonblocking property of $G_1 \parallel \cdots \parallel G_n$.

Proof Let

$$\tilde{C} : \tilde{y}^0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} \tilde{y}^k \quad \text{with } \tilde{y}^i = (\tilde{x}_1^i, \ldots, \tilde{x}_n^i) \text{ for } i = 0, \ldots, k \ ; \tag{13}$$

be the reduced path that results from the loop on lines 1–3, as per line 4. The loop on lines 1–3 modifies an abstract counterexample to the nonblocking property of $\tilde{G}_1 \| G_2 \| \cdots \| G_n$ by removing τ-transitions performed by $\tilde{G}_1$ whose source and target state components of $G_2, \ldots, G_n$ must be equal. Then the modified $\tilde{C}$ continues to be accepted by $G_2, \ldots, G_n$ (but not necessarily by $\tilde{G}_1$), and the end state of $\tilde{C}$ is unchanged as the loop removes transitions with their source states. Thus,

$$\tilde{C} \text{ is accepted by } G_2 \parallel \cdots \parallel G_n \ ; \tag{14}$$

$$\mathcal{L}^\omega(\tilde{y}^k) = \emptyset \ . \tag{15}$$

It is first shown that the main loop of Algorithm 1 satisfies the following *loop invariant*. At the beginning of every iteration, on line 11, for every pair $(C, c) \in$ *Queue* with

$$C : y^0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} y^m \quad \text{with } y^i = (x_1^i, \ldots, x_n^i) \text{ for } i = 0, \ldots, m \ ; \tag{16}$$

the following conditions hold:

(i) $c \leq k$;
(ii) C is accepted by $G_1 \parallel \cdots \parallel G_n$;
(iii) $(x_2^m, \ldots, x_n^m) = (\tilde{x}_2^c, \ldots, \tilde{x}_n^c)$;

First consider the initial pairs $(C, 0)$ added to the *Queue* on line 7, i.e., $C = y^0 = (x_1^\circ, \tilde{x}_2^0, \ldots, \tilde{x}_n^0)$ is a one-state path. Obviously $0 \leq k$, so (i) holds. Following (14), it is clear that $\tilde{x}_j^0 \in Q_j^\circ$ for $j = 2, \ldots, n$, and as $x_1^\circ \in Q_1^\circ$ from line 5, it follows that $y^0 = (x_1^\circ, \tilde{x}_2^0, \ldots, \tilde{x}_n^0)$ is an initial state of $G = G_1 \parallel \cdots \parallel G_n$, showing (ii). Also $(x_2^m, \ldots, x_n^m) = (x_2^0, \ldots, x_n^0) = (\tilde{x}_2^0, \ldots, \tilde{x}_n^0)$ from line 6, showing (iii).

Now assume a pair (C, c) that satisfies (i)–(iii) is removed from the *Queue* on line 11, and consider the pairs added during the loop on line 21, 27, or 33.

If $(C', c+1)$ is added on line 21, then $c < k$ from (i) and line 13, and thus $c+1 \leq k$ which shows (i). Furthermore, $\sigma_{c+1} \in \Sigma_1$ from line 17, and $x_1^m \xrightarrow{\sigma_{c+1}}_1 z_1$ in G_1 from line 18, and $C' = C \xrightarrow{\sigma_{c+1}} (z_1, \tilde{x}_2^{c+1}, \ldots, \tilde{x}_n^{c+1})$ from line 20. The end state of C is $y^m = (x_1^m, \ldots, x_n^m) = (x_1^m, \tilde{x}_2^c, \ldots, \tilde{x}_n^c)$ as C satisfies (iii), and given (14) it holds that $(x_1^m, \tilde{x}_2^c, \ldots, \tilde{x}_n^c) \xrightarrow{\sigma_{c+1}} (z_1, \tilde{x}_2^{c+1}, \ldots, \tilde{x}_n^{c+1})$ in $\tilde{G}_1 \| G_2 \| \cdots \| G_n$. As C is accepted by $G_1 \parallel \cdots \parallel G_n$ according to (ii), it follows that C' also has this property. Also the end state components for $G_2, \ldots, G_n$ in C' are $\tilde{x}_2^{c+1}, \ldots, \tilde{x}_n^{c+1}$ from line 20, which shows (iii).

If $(C', c+1)$ is added on line 27, then $c < k$ from (i) and line 13, and thus $c+1 \leq k$ which shows (i). Furthermore, $\sigma_{c+1} \notin \Sigma_1$ from line 17 and $C' = C \xrightarrow{\sigma_{c+1}}$

$(x_1^m, \tilde{x}_2^{c+1}, \ldots, \tilde{x}_n^{c+1})$ from line 26. The end state of C is $y^m = (x_1^m, \ldots, x_n^m) = (x_1^m, \tilde{x}_2^c, \ldots, \tilde{x}_n^c)$ as C satisfies (iii), and from (14) and $\sigma_{c+1} \notin \Sigma_1$ it follows that $(x_1^m, \tilde{x}_2^c, \ldots, \tilde{x}_n^c) \xrightarrow{\sigma_{c+1}} (x_1^m, \tilde{x}_2^{c+1}, \ldots, \tilde{x}_n^{c+1})$. As C is accepted by $G_1 \| \cdots \| G_n$ according to (ii), it follows that C' also has this property. Also the end state components for $G_2, \ldots, G_n$ in C' are $\tilde{x}_2^{c+1}, \ldots, \tilde{x}_n^{c+1}$ from line 26, which shows (iii).

If (C', c) is added on line 33, then (i) continues to hold as c is unchanged. Further, $x_1^m \xrightarrow{\tau}_1 z_1$ from line 30 and $C' = C \xrightarrow{\tau} (z_1, \tilde{x}_2^c, \ldots, \tilde{x}_n^c)$ from line 32. Then $(x_1^m, \tilde{x}_2^c, \ldots, \tilde{x}_n^c) \xrightarrow{\tau} (z_1, \tilde{x}_2^c, \ldots, \tilde{x}_n^c)$, and as C is accepted by $G_1 \| \cdots \| G_n$ according to (ii), it is clear that C' also has this property. The end state components for $G_2, \ldots, G_n$ in C' are $\tilde{x}_2^c, \ldots, \tilde{x}_n^c$, which shows (iii).

This completes the proof of the loop invariant and shows that (i)–(iii) hold for all pairs $(C, c) \in \mathit{Queue}$. Now assume C is returned from line 15. Then C has been removed from the *Queue* on line 11, so there exists c such that (C, c) satisfies (i)–(iii). Then $c = k$ from line 13 and $\mathcal{L}^\omega(x_1^m) \subseteq \mathcal{L}^\omega(\tilde{x}_1^k)$ from line 14, where the end state of C is $y^m = (x_1^m, \ldots, x_n^m)$. Firstly, C is accepted by $G_1 \| \cdots \| G_n$ because of (ii), and secondly it follows from (iii) that $y^m = (x_1^m, \ldots, x_n^m) = (x_1^m, \tilde{x}_2^c, \ldots, \tilde{x}_n^c) = (x_1^m, \tilde{x}_2^k, \ldots, \tilde{x}_n^k)$, which implies $\mathcal{L}^\omega(y^m) = \mathcal{L}^\omega(x_1^m) \cap \mathcal{L}^\omega(\tilde{x}_2^k) \cap \cdots \cap \mathcal{L}^\omega(\tilde{x}_n^k) \subseteq \mathcal{L}^\omega(\tilde{x}_1^k) \cap \mathcal{L}^\omega(\tilde{x}_2^k) \cap \cdots \cap \mathcal{L}^\omega(\tilde{x}_n^k) = \mathcal{L}^\omega(\tilde{y}^k) = \emptyset$ by (15). This shows that C is a counterexample to the nonblocking property of $G_1 \| \cdots \| G_n$ according to Def. 4. □

Having established partial correctness, the next step towards the correctness proof of Algorithm 1 is to prove termination. The next Lemma 3 proves that the main loop on lines 10–37 is guaranteed to terminate, either by returning a result from line 15 or by the loop entry condition of a non-empty *Queue* on line 10 becoming false. The second case is not desirable, because the algorithm does not return any result in this case, and it will be ruled out by Lemma 4 below. The proof of Lemma 3 still does not require the assumption of the counterexample-based preorder. Thus, the algorithm terminates under all circumstances, but the result may be inconclusive if $\tilde{G}_1 \lesssim_{\mathrm{ce}} G_1$ does not hold.

Lemma 3 Assume that Algorithm 1 is run with a counterexample to the nonblocking property of $\tilde{G}_1 \| G_2 \| \cdots \| G_n$ as input. Then the main loop on lines 10–37 will terminate after a finite number of iterations.

Proof Denote by

$$s_i = \sigma_1 \cdots \sigma_i \qquad \text{for } i = 0, \ldots, k \tag{17}$$

the strings of events on the first i transitions on the reduced path $\tilde{C}$ (13) resulting from the loop on lines 1–3. Based on this, the set of pairs to be visited by Algorithm 1 is defined as

$$\mathbf{V} = \{\, (v_1, c) \in Q_1 \times \{0, \ldots, k\} \mid G_1 \overset{P(s_c)}{\Longrightarrow} v_1 \,\} \,. \tag{18}$$

The set $\mathbf{V}$ contains pairs of states v_1 of G_1 and numbers c of processed events from $\tilde{C}$, which are expected to be added to the set *Visited*.

It is now shown that the main loop of Algorithm 1 satisfies the following loop invariant conditions in addition to (i)–(iii) that were shown in Lemma 2. At the beginning of every iteration, on line 11, for every pair $(C, c) \in \mathit{Queue}$ and for the set *Visited* the following conditions hold:

(iv) $P(\mathrm{trace}(C)) = P(s_c)$;

(v) *Visited* $\subseteq \mathbf{V}$.

First, if a pair $(C, 0)$ is added to the *Queue* on line 7, then $\text{trace}(C) = \varepsilon$ and thus $P(\text{trace}(C)) = P(\varepsilon) = P(s_0)$ using (17), which shows (iv). Also, if $(x_1^\circ, 0)$ is added to *Visited* on line 8, then $x_1^\circ \in Q_1^\circ$ so that $G_1 \stackrel{\varepsilon}{\Rightarrow} x_1^\circ$ with $s_0 = \varepsilon$, which implies $(x_1^\circ, 0) \in \mathbf{V}$ using (18) and shows (v).

Now assume the main loop enters an iteration where (i)–(v) hold at the beginning, and a pair (C, c) that satisfies (i)–(iv) is removed from the *Queue* on line 11. Note that by (ii) and (iv)

$$G_1 \stackrel{P(s_c)}{\Longrightarrow} x_1^m \ , \tag{19}$$

where x_1^m is the state component of G_1 at the end of C. Consider the pairs added on lines 21–22, 27–28, and 33–34.

If $(C', c+1)$ is added on line 21, then $c < k$ from (i) and line 13, and $\sigma_{c+1} \in \Sigma_1$ from line 17, and $x_1^m \xrightarrow{\sigma_{c+1}}_1 z_1$ in G_1 from line 18, and $\text{trace}(C') = \text{trace}(C)\sigma_{c+1}$ from line 20. Then, since (iv) holds for (C, c), it follows that $P(\text{trace}(C')) = P(\text{trace}(C)\sigma_{c+1}) = P(s_c\sigma_{c+1}) = P(s_{c+1})$, showing (iv) for $(C', c+1)$. If next $(z_1, c+1)$ is added to *Visited* on line 22, then $G_1 \stackrel{P(s_c)}{\Longrightarrow} x_1^m \xrightarrow{\sigma_{c+1}} z_1$ by (19), and with $P(s_c\sigma_{c+1}) = P(s_{c+1})$ it is clear that $G_1 \stackrel{P(s_{c+1})}{\Longrightarrow} z_1$. This means $(z_1, c+1) \in \mathbf{V}$ by (18) and shows that (v) continues to hold.

If $(C', c+1)$ is added on line 27, then $c < k$ from (i) and line 13, and $\sigma_{c+1} \notin \Sigma_1$ from line 17, and $\text{trace}(C') = \text{trace}(C)\sigma_{c+1}$ from line 26. It is shown in the same way as in the case of lines 21–22 above that (iv) and (v) continue to hold, just using x_1^m instead of z_1 and noting that $x_1^m \xrightarrow{\sigma_{c+1}}_1 x_1^m$ for $\sigma_{c+1} \notin \Sigma_1$.

If (C', c) is added on line 33, then $x_1^m \xrightarrow{\tau}_1 z_1$ from line 30 and $\text{trace}(C') = \text{trace}(C)\tau$ from line 32. Then $P(\text{trace}(C')) = P(\text{trace}(C)\tau) = P(\text{trace}(C)) = P(s_c)$ as (C, c) satisfies (iv), which then also holds for (C', c). If next (z_1, c) is added to *Visited* on line 34, then $G_1 \stackrel{P(s_c)}{\Longrightarrow} x_1^m \xrightarrow{\tau} z_1$ by (19) and thus $G_1 \stackrel{P(s_c)}{\Longrightarrow} z_1$. This means $(z_1, c) \in \mathbf{V}$ by (18) and shows that (v) continues to hold.

This completes the proof of the additional invariants. In particular, (v) *Visited* $\subseteq$ $\mathbf{V}$ holds before and after each iteration. In Algorithm 1, whenever a pair (C, c) is added to the *Queue* on line 7, 21, 27, or 33, then the following instruction also adds an entry to *Visited* that is not already contained. As *Visited* $\subseteq \mathbf{V}$ and $\mathbf{V}$ is a finite set with at most $|Q_1| \cdot (k+1)$ elements, it follows that only a finite number of pairs can be added to the *Queue*. As each iteration of the main loop removes one element from the *Queue* on line 11, the number of iterations must be finite. □

Lemma 3 shows that the main loop on lines 10–37 of Algorithm 1 terminates. The other loops on lines 1–3, 5–9, 18–24, and 30–36 clearly have finite numbers of $|\tilde{C}|$ or $|Q_1|$ iterations, so this is enough to prove termination of Algorithm 1.

Additionally, the proof of Lemma 3 gives the number of iterations of the main loop as $|Q_1| \cdot (k+1)$ where $k = |\tilde{C}|$ is the number of events in the reduced abstract counterexample. This gives rise to a time complexity estimation by considering that each iteration of the main loop may also include the loops on lines 18–24 and 30–36 which may visit one transition to each state of the nondeterministic FSM G_1, i.e., $|Q_1|$ iterations. The initial loops on lines 1–3 and 5–9 are dominated by the main loop. This gives the worst-case time complexity of $O(|\tilde{C}||Q_1|^2)$. This estimate is based on the assumption that the lookup of pairs in *Visited* on lines 19,

25, and 31 and the construction of traces C' on lines 20, 26, and 32 are done in constant time. This can be achieved by representing *Visited* as a hash set and the traces C' as linked lists, for example.

It remains to be shown that Algorithm 1 does not only terminate, but that it terminates by returning a result from line 15. This is only guaranteed under the assumption $\tilde{G}_1 \lesssim_{\mathrm{ce}} G_1$ of the counterexample-based preorder.

Lemma 4 Let the input of Algorithm 1 be such that $\tilde{C}$ is a counterexample to the nonblocking property of $\tilde{G}_1 \parallel G_2 \parallel \cdots \parallel G_n$ and $\tilde{G}_1 \lesssim_{\mathrm{ce}} G_1$. Then the main loop on lines 10–37 terminates by returning a result from line 15.

Proof It follows from Lemma 3 that the main loop terminates either by returning a result or because of an empty *Queue*. Now assume that the main loop never returns from line 15. It will be shown that this results in a situation where the loop must reach line 15, i.e., a contradiction.

First, it can be observed from Algorithm 1 that, whenever a pair (v_1, c) is added to *Visited* (on line 8, 22, 28, or 34), then the preceding line adds a pair (C, c) to the *Queue* where the state component of G_1 in the last state of C is v_1.

It is now shown that Algorithm 1 will eventually add every element of $\mathbf{V}$, defined by (18), to *Visited*. Consider $(v_1, c) \in \mathbf{V}$. Then $0 \leq c \leq k$ and $G_1 \overset{P(s_c)}{\Longrightarrow} v_1$, which means that there exists a trace $t \in \Sigma_\tau^*$ with $P(t) = P(s_c)$ such that $G_1 \overset{t}{\to} v_1$. The claim is shown by induction on $|t| + c$.

For the inductive base, $|t| + c = 0$ and thus $t = \varepsilon$ and $c = 0$. Then $G_1 \overset{t}{\to} v_1$ implies $v_1 \in Q_1^\circ$, so that $(v_1, c) = (v_1, 0)$ is added to *Visited* on line 8.

For the inductive assumption, assume the claim has been shown for some $N \geq 0$. That is, for all $(v_1, c) \in \mathbf{V}$ and all $t \in \Sigma_\tau^*$ such that $P(t) = P(s_c)$ and $G_1 \overset{t}{\to} v_1$, if $|t| + c \leq N$, then (v_1, c) is added to *Visited* by Algorithm 1.

For the inductive step, consider $(v_1, c) \in \mathbf{V}$ and $t \in \Sigma_\tau^*$ such that $P(t) = P(s_c)$ and $G_1 \overset{t}{\to} v_1$ and $|t| + c = N + 1$. Note that $|t| + c > 0$ so that $|t| > 0$ or $c > 0$. Thus, $t \neq \varepsilon$ or $s_c \neq \varepsilon$. Then at least one of the following holds: t ends with τ, or s_c ends with τ, or given $P(t) = P(s_c)$ both t and s_c end with the same event $\sigma \neq \tau$. These three cases are considered separately in the following.

Case 1: $t = u\tau$. Then $G_1 \overset{u}{\to} w_1 \overset{\tau}{\to} v_1$ and $P(u) = P(u\tau) = P(t) = P(s_c)$, which means $G_1 \overset{P(s_c)}{\Longrightarrow} w_1$ and thus $(w_1, c) \in \mathbf{V}$. As also $|u| + c = |t| - 1 + c = N$, it follows by inductive assumption that (w_1, c) is added to *Visited* by Algorithm 1. Then also a pair (C, c) is added to the *Queue* where the state component of G_1 in the last state of C is w_1. If the algorithm does not terminate early, this pair (C, c) is removed from the *Queue* on line 11 during some later iteration, at which point $x_1^m = w_1$. Then the loop on lines 30–36 must process the transition $x_1^m = w_1 \overset{\tau}{\to} v_1$, which results in addition of (v_1, c) to *Visited* on line 34 (unless this pair is already contained beforehand).

Case 2: $s_c = s_{c-1}\sigma_c$ with $\sigma_c = \tau$. Then $P(t) = P(s_c) = P(s_{c-1}\tau) = P(s_{c-1})$, which given $G_1 \overset{t}{\to} v_1$ means $G_1 \overset{P(s_{c-1})}{\Longrightarrow} v_1$ and thus $(v_1, c-1) \in \mathbf{V}$. As also $|t|+c-1 = N$, it follows by inductive assumption that $(v_1, c-1)$ is added to *Visited*. Then also a pair $(C, c-1)$ is added to the *Queue* where the state component of G_1 in the last state of C is v_1. This pair is removed on line 11 during some later iteration, at which point $x_1^m = v_1$ and the value of c is $c - 1$. As $c \leq k$ for $(v_1, c) \in \mathbf{V}$ it is clear that $c - 1 \neq k$ on line 13, and $\sigma_{(c-1)+1} = \sigma_c = \tau \notin \Sigma_1$ on line 17. Then the

code on lines 26–28 gets executed, so $(x_1^m, (c-1)+1) = (v_1, c)$ is added to *Visited* on line 28.

Case 3: $s_c = s_{c-1}\sigma_c$ and $t = u\sigma_c$ with $\sigma_c \neq \tau$. Then $G_1 \stackrel{u}{\to} w_1 \stackrel{\sigma_c}{\to} v_1$ and $P(u) = P(s_{c-1})$, which means $G_1 \stackrel{P(s_{c-1})}{\Longrightarrow} w_1$ and thus $(w_1, c-1) \in \mathbf{V}$. As also $|u| + (c-1) = |t| - 1 + c - 1 = (N+1) - 2 < N$, it follows by inductive assumption that $(w_1, c-1)$ is added to *Visited*. Then also a pair $(C, c-1)$ is added to the *Queue* where the state component of G_1 in the last state of C is w_1. This pair is removed on line 11 during some later iteration, at which point $x_1^m = w_1$ and the value of c is $c-1$. As $c \leq k$ it is clear that $c-1 \neq k$ and $\sigma_{(c-1)+1} = \sigma_c$. Consider two further cases. If $\sigma_c \in \Sigma_1$, then as also $c-1 \neq k$, the loop on lines 18–24 gets executed. Given $x_1^m = w_1 \stackrel{\sigma_c}{\to} v_1$ it follows that the pair $(v_1, (c-1)+1) = (v_1, c)$ is added to *Visited* on line 22. Otherwise, if $\sigma_c \notin \Sigma_1$, the code on lines 26–28 gets executed, and it follows from $w_1 \stackrel{\sigma_c}{\to} v_1$ that $w_1 = v_1$. Therefore, $(x_1^m, (c-1)+1) = (w_1, c) = (v_1, c)$ is added to *Visited* on line 28.

This completes the induction and shows that every pair from $\mathbf{V}$ gets added to the set *Visited* at some point. Now recall that the input to Algorithm 1 is a counterexample to the nonblocking property of $\tilde{G}_1 \parallel G_2 \parallel \cdots \parallel G_n$, which is modified through the loop on lines 1–3 by deletion of τ-transitions. The projection $P(\mathrm{trace}(\tilde{C})) = P(s_k)$ on line 4 is unchanged from the original input, so that $\tilde{G}_1 \stackrel{P(s_k)}{\Longrightarrow} \tilde{x}_1^k$ where $\tilde{x}_1^k$ is the state component of $\tilde{G}_1$ at the end of a concrete counterexample. As $\tilde{G}_1 \lesssim_{\mathrm{ce}} G_1$, by Def. 8 there exists a state $x_1 \in Q_1$ such that $G_1 \stackrel{P(s_k)}{\Longrightarrow} x_1$ and $\mathcal{L}^\omega(x_1) \subseteq \mathcal{L}^\omega(\tilde{x}_1^k)$. Then $(x_1, k) \in \mathbf{V}$ by (18), and this pair gets added to *Visited* at some point during execution of Algorithm 1. Then also a pair (C, k) is added to the *Queue* where the state component of G_1 in the last state of C is x_1. This pair is removed on line 11 during some later iteration, at which point $c = k$ and $x_1^m = x_1$. Thus, $c = k$ and $\mathcal{L}^\omega(x_1^m) = \mathcal{L}^\omega(x_1) \subseteq \mathcal{L}(\tilde{x}_1^k)$, so Algorithm 1 returns C from line 15. □

The following Prop. 5 combines the results of the above lemmas and confirms the *total correctness* of Algorithm 1. That is, under the assumption $\tilde{G}_1 \lesssim_{\mathrm{ce}} G_1$ of the counterexample-based preorder, the algorithm terminates and gives a correct result.

Proposition 5 Let the input of Algorithm 1 be such that $\tilde{C}$ is a counterexample to the nonblocking property of $\tilde{G}_1 \parallel G_2 \parallel \cdots \parallel G_n$ and $\tilde{G}_1 \lesssim_{\mathrm{ce}} G_1$. Then Algorithm 1 terminates after a finite number of steps and returns a counterexample to the nonblocking property of $G_1 \parallel \cdots \parallel G_n$.

Proof By Lemma 4, the algorithm terminates by returning a result from line 15. By Lemma 2, this result is a counterexample to the nonblocking property of $G_1 \parallel \cdots \parallel G_n$. □

3.4 Specific Abstraction Rules

In order to obtain a concrete counterexample from Algorithm 1, it must be guaranteed that the abstraction does not only preserve conflict equivalence but also the stronger counterexample-based preorder of Def. 8. Fortunately, many conflict-preserving abstraction rules [7, 21] can be shown to fit in this framework. This section presents the relevant results for the abstraction rules of [7].

For an abstraction to satisfy Def. 8, it must be shown that for every state reachable in the abstraction there exists a state in the original FSM, which is reached by the same trace and has a smaller accepting language. As state merging abstractions are constructed as quotient FSMs, it is noticed that the accepting language of a state x is always contained in the accepting language of its equivalence class $[x]$ in a quotient FSM. This is well-known for all state merging abstractions [2].

Lemma 6 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $G/\!\sim$ be its quotient modulo an equivalence relation $\sim \subseteq Q \times Q$. Then $\mathcal{L}^\omega(x) \subseteq \mathcal{L}^\omega([x])$ for every state $x \in Q$.

Proof Let $x \in Q$ and $s \in \mathcal{L}^\omega(x)$. Then there exists a path $x = x_0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_n} x_n \xrightarrow{\omega}$ in G where $s = P(\sigma_1 \cdots \sigma_n)$. Then by Def. 7, $[x] = [x_0] \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_n} [x_n] \xrightarrow{\omega}$ is a path in $G/\!\sim$, and this is enough to show $s \in \mathcal{L}^\omega([x])$. □

Based on Lemma 6, for every abstract state $[x]$ it is clear that there exists $x \in [x]$ with a smaller marked language. For counterexample expansion, it remains to be shown that these states can also be reached with the same traces. This second condition is not true for all quotient FSMs, and it depends on the specific equivalence relation used. One equivalence that can be used is the well-known *weak bisimulation* or *observation equivalence* [19].

Definition 9 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM. An equivalence relation $\approx \subseteq Q \times Q$ is a *weak bisimulation* or *observation equivalence* relation on G if for all states $x_1, x_2 \in Q$ such that $x_1 \approx x_2$ the following condition holds: if $x_1 \xrightarrow{\sigma} y_1$ for some $y_1 \in Q$ and $\sigma \in \Sigma_{\tau,\omega}$, then there exists $y_2 \in Q$ such that $y_1 \approx y_2$ and $x_2 \overset{P(\sigma)}{\Longrightarrow} y_2$.

Two states are observation equivalent if they have got exactly the same enabled events, leading to equivalent successor states. This property of observation equivalence ensures that Lemma 6 can be reversed to construct paths in the original FSM G from paths in its abstraction $G/\!\approx$. The proof of the following lemma is by straightforward induction from Def. 9.

Lemma 7 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\approx \subseteq Q \times Q$ be an observation equivalence relation on G. If $\tilde{x} \overset{s}{\Rightarrow} \tilde{y}$ in $G/\!\approx$, then for every state $x \in \tilde{x}$ there exists a state $y \in \tilde{y}$ such that $x \overset{s}{\Rightarrow} y$ in G.

Observation equivalence is a well-known equivalence with efficient algorithms that preserves conflict equivalence [7]. It is now shown that a quotient FSM modulo an observation equivalence satisfies Def. 8 so that Algorithm 1 can be used for counterexample expansion.

Proposition 8 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\approx \subseteq Q \times Q$ be an observation equivalence relation on G. Then $G/\!\approx \lesssim_{\text{ce}} G$.

Proof Let $G/\!\approx \overset{s}{\Rightarrow} \tilde{x}$ for some $s \in \Sigma^*$ and $\tilde{x} \in Q/\!\approx$. Then there exists an initial state class $\tilde{x}^\circ \in \tilde{Q}^\circ$ such that $\tilde{x}^\circ \overset{s}{\Rightarrow} \tilde{x}$. By Def. 7, there exists an initial state $x^\circ \in Q^\circ \cap \tilde{x}^\circ$. By Lemma 7, it follows that there exists a state $x \in \tilde{x}$ such that $x^\circ \overset{s}{\Rightarrow} x$, i.e., $G \overset{s}{\Rightarrow} x$. By Lemma 6, it is clear that $\mathcal{L}^\omega(x) \subseteq \mathcal{L}^\omega(\tilde{x})$, and this is enough to show $G \lesssim_{\text{ce}} G/\!\approx$ using Def. 8. □

Observation equivalence is the most effective known abstraction rule for compositional nonblocking verification in terms of state-space reduction [7,21]. This is known as the *Observation Equivalence Rule* [7]. Its use is justified by the fact that observation equivalence implies conflict equivalence, which is proved in a more general setting in [16]. The following proof is more direct and demonstrates the use of the counterexample-based preorder.

Proposition 9 (Observation Equivalence Rule) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\approx \subseteq Q \times Q$ be an observation equivalence relation on G. Then $G \simeq_{\text{conf}} G/\approx$.

Proof By Props. 1 and 8, it is clear that $G/\approx \lesssim_{\text{conf}} G$. It is now shown that also $G \lesssim_{\text{ce}} G/\approx$ as this implies $G \lesssim_{\text{conf}} G/\approx$ by Prop. 1. Let $G \stackrel{s}{\Rightarrow} x$ for some $s \in \Sigma^*$. It follows from Def. 7 that $G/\approx \stackrel{s}{\Rightarrow} [x]$, so it remains to be shown that $\mathcal{L}^\omega([x]) \subseteq \mathcal{L}^\omega(x)$. Let $s \in \mathcal{L}^\omega([x])$. Then $[x] \stackrel{s\omega}{\Rightarrow} \tilde{y}$ in $G/\approx$. By Lemma 7, it follows that there exists $y \in \tilde{y}$ such that $x \stackrel{s\omega}{\Rightarrow} y$ in G, which means that $s \in \mathcal{L}^\omega(x)$. □

Conflict equivalence allows for other means of simplification beyond observation equivalence. The compositional verification method [7] includes abstraction rules where the quotient is based on a different kind of equivalence relation, called incoming equivalence.

Definition 10 [7] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM. An equivalence relation $\sim_{\text{inc}} \subseteq Q \times Q$ is an *incoming equivalence* relation on G if, for all states $x_1, x_2 \in Q$ such that $x_1 \sim_{\text{inc}} x_2$ the following conditions hold.

(i) If $Q^\circ \stackrel{\varepsilon}{\Rightarrow} x_1$ then $Q^\circ \stackrel{\varepsilon}{\Rightarrow} x_2$.
(ii) If $x \stackrel{\sigma}{\Rightarrow} x_1$ for some $x \in Q$ and $\sigma \in \Sigma_\omega$, then it also holds that $x \stackrel{\sigma}{\Rightarrow} x_2$.

Two states are incoming equivalent if they have got the same incoming transitions from exactly the same source states. Incoming equivalent states typically result from nondeterministic branching. If a state in a nondeterministic automaton has two successor states reached by the same event, then these successor states are incoming equivalent provided that they do not have other incoming transitions. Incoming equivalence also allows the construction of paths in the original FSM from paths in its abstraction, making it another sufficient condition for Def. 8.

Lemma 10 [7] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim_{\text{inc}} \subseteq Q \times Q$ be an incoming equivalence relation on G. If $G/\sim_{\text{inc}} \stackrel{s}{\Rightarrow} \tilde{x}$, then there exists a state $x \in \tilde{x}$ such that $G \stackrel{s}{\Rightarrow} x$.

Proposition 11 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim_{\text{inc}} \subseteq Q \times Q$ be an incoming equivalence relation on G. Then $G/\sim_{\text{inc}} \lesssim_{\text{ce}} G$.

Proof Let $G/\sim_{\text{inc}} \stackrel{s}{\Rightarrow} \tilde{x}$ for some $s \in \Sigma^*$ and $\tilde{x} \in Q/\sim_{\text{inc}}$. By Lemma 10, there exists a state $x \in \tilde{x}$ such that $G \stackrel{s}{\Rightarrow} x$. By Lemma 6, it is clear that $\mathcal{L}^\omega(x) \subseteq \mathcal{L}^\omega(\tilde{x})$, and this is enough to show $G/\sim_{\text{inc}} \lesssim_{\text{ce}} G$ using Def. 8. □

By Prop. 11, every quotient FSM constructed with an incoming equivalence relation results in an abstraction that is counterexample-based less conflicting than the original FSM. Incoming equivalence alone does not ensure the converse,

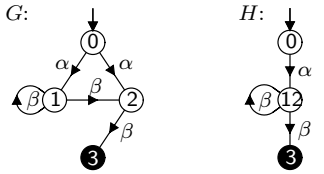

Fig. 2 Example of application of the Active Events Rule.

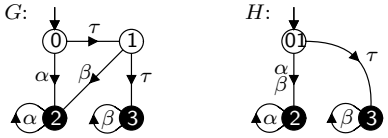

Fig. 3 Example of application of the Silent Continuation Rule.

i.e, that the abstraction is also more conflicting than the original, so additional conditions are imposed to ensure a conflict equivalent abstraction.

One of these is described as the *Active Events Rule* [7]. In addition to being incoming equivalent, two states must also have exactly the same enabled events, either immediately or after some τ-transitions. Merging states that are equivalent subject to this combined condition preserves conflict equivalence.

Proposition 12 (Active Events Rule [7]) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim \subseteq Q \times Q$ be an incoming equivalence relation on G such that, for all $x_1, x_2 \in Q$ with $x_1 \sim x_2$ and all events $\sigma \in \Sigma_\omega$ it holds that $x_1 \stackrel{\sigma}{\Rightarrow}$ if and only if $x_2 \stackrel{\sigma}{\Rightarrow}$. Then $G \simeq_{\text{conf}} G/\!\sim$.

Example 3 [7] In Fig. 2, states 1 and 2 in G have incoming transitions from 0 associated with α and from 1 associated with β, which establishes incoming equivalence. Furthermore, the only event enabled from these states is β. According to Prop. 12, these two states are equivalent and can be merged into a single state 12 as shown in H.

Another abstraction based on incoming equivalence is the *Silent Continuation Rule* [7]. It allows for the merging of incoming equivalent states, which in addition have outgoing τ-transitions that eventually lead to *stable* states, i.e., states with no further outgoing τ-transitions.

Proposition 13 (Silent Continuation Rule [7]) Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an FSM, and let $\sim \subseteq Q \times Q$ be an incoming equivalence relation on G such that, for all $x_1, x_2 \in Q$ with $x_1 \sim x_2$ it holds that either $x_1 = x_2$ or there exist states $y_1, y_2 \in Q$ such that $x_1 \neq y_1$ and $x_2 \neq y_2$ and $x_1 \stackrel{\varepsilon}{\Rightarrow} y_1 \not\stackrel{\tau}{\rightarrow}$ and $x_2 \stackrel{\varepsilon}{\Rightarrow} y_2 \not\stackrel{\tau}{\rightarrow}$. Then $G \simeq_{\text{conf}} G/\!\sim$.

Example 4 [7] In Fig. 3, states 0 and 1 in G are both considered initial as they both can be reached silently from the initial state 0. This is enough to satisfy incoming equivalence in this case, since neither state is reachable by any event other than τ. Moreover, both states can, by executing at least one silent transition, reach the stable state 3, which has no outgoing τ transitions. By Prop. 13, states 0 and 1 in G are conflict equivalent and can be merged into state 01 as shown in H.

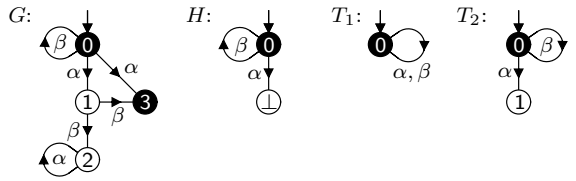

Fig. 4 Example of certain conflicts abstraction.

As incoming equivalence results in a counterexample-based less conflicting abstraction by Prop. 11, and then also in a less conflicting abstraction by Prop. 1, to complete the proofs of Props. 12 and 13 it only needs to be shown that these abstractions are also more conflicting than the original FSMs. These proofs are not given here, since full proofs of conflict equivalence appear in [7].

By inspecting the proofs of Props. 8 and 11, it is furthermore noticed that the end state x of the concrete counterexample always is a member of the class $\tilde{x}$ that represents the end state of the abstract counterexample, $x \in \tilde{x}$. Here it is guaranteed by Lemma 6 that $\mathcal{L}^\omega(x) \subseteq \mathcal{L}^\omega(\tilde{x})$. This means that the language containment check $\mathcal{L}^\omega(x_1^i) \subseteq \mathcal{L}^\omega(\tilde{x}_1^k)$ on line 14 of Algorithm 1 can be replaced by simply checking whether $x_1^i \in \tilde{x}_1^k$. Such a state is a guaranteed to exist according to the proofs of Props. 8 and 11, and unlike language containment, the state membership check can be completed in constant time.

Two further abstraction rules are known as the *Only Silent Incoming Rule* and the *Only Silent Outgoing Rule* [7]. These rules can be described as combinations of the Observation Equivalence and Silent Continuation Rules, so the results of this section apply to these rules also. This means that Algorithm 1 can be used to compute counterexamples for all the abstraction rules proposed in [7]—with the exception of the *Certain Conflicts Rule*, which requires a different approach and is considered in the next subsection.

3.5 Certain Conflicts

The converse of Prop. 1 does not hold, so being counterexample-based less conflicting is a strictly stronger property than being less conflicting. An example is abstraction by *certain conflicts* [13, 14].

Example 5 FSMs G and H in Fig. 4 are conflict equivalent, because any FSM T that can initially execute α is conflicting with both G and H. Note that execution of α may take G to state $\mathsf{1}$, where β is needed to reach an accepting state, but β also leads to the blocking state $\mathsf{2}$. However, H is not counterexample-based less conflicting than G, because for $H \stackrel{\alpha}{\Rightarrow} \bot$ with $\mathcal{L}^\omega(\bot) = \emptyset$, there is no state in G reachable via α with a smaller accepting language.

Counterexample expansion is more difficult in the absence of Def. 8. Assume that the remainder of the system in Fig. 4 behaves like T_1, and the abstract counterexample for $H \,\|\, T_1$ is $(\mathsf{0},\mathsf{0}) \stackrel{\alpha}{\to} (\bot,\mathsf{0})$. Attempts to convert this to a concrete counterexample with end state $(\mathsf{1},\mathsf{0})$ or $(\mathsf{3},\mathsf{0})$ in $G \,\|\, T_1$ fail as these are not blocking states. A concrete counterexample can only be obtained by *extension*, e.g., $(\mathsf{0},\mathsf{0}) \stackrel{\alpha}{\to} (\mathsf{1},\mathsf{0}) \stackrel{\beta}{\to} (\mathsf{2},\mathsf{0})$ is a counterexample to the nonblocking property of $G \,\|\, T_1$. How to

extend does not only depend on the abstracted FSM but also on the rest of the system [13]. An abstract counterexample for $H \parallel T_2$ in Fig. 4 is $(0,0) \xrightarrow{\alpha} (\bot,1)$, which cannot and does not need to be extended.

In the following, a variant of the of the *Certain Conflicts Rule* [7] is considered, which is exemplary for all cases where Def. 8 does not apply.

Definition 11 (Limited Certain Conflicts Rule) Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an FSM. Define sets of *limited certain conflict states* inductively:

$$\mathrm{lcc}_G^0 = \{\, x \in Q \mid \mathcal{L}^\omega(x) = \emptyset \,\} \;; \tag{20}$$

$$\mathrm{lcc}_G^{i+1} = \{\, x \in Q \mid \text{for every path } x = x_0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} x_k \xrightarrow{\omega} \text{ there exists } j \geq 0 \text{ such that } j \leq k \text{ and } x_j \overset{\varepsilon}{\Rightarrow} \mathrm{lcc}_G^i, \text{ or } j < k \text{ and } x_j \overset{\sigma_{j+1}}{\Longrightarrow} \mathrm{lcc}_G^i \,\} \;; \tag{21}$$

$$\mathrm{lcc}_G = \bigcup_{i \geq 0} \mathrm{lcc}_G^i \;. \tag{22}$$

The *limited certain conflicts abstraction* of G is $\textsc{Lcc}(G) = \langle \Sigma, Q_{\mathrm{lcc}}, \to_{\mathrm{lcc}}, Q^\circ_{\mathrm{lcc}} \rangle$ where $Q_{\mathrm{lcc}} = (Q \setminus \mathrm{lcc}_G) \cup \{\bot\}$ (with $\bot \notin Q$); $x \xrightarrow{\sigma}_{\mathrm{lcc}} y$ if $x, y \neq \bot$ and $x \xrightarrow{\sigma} y$ and $x \overset{P(\sigma)}{\Longrightarrow} \mathrm{lcc}_G$ does not hold, or $x \neq \bot = y$ and $x \xrightarrow{\sigma} \mathrm{lcc}_G$; and $Q^\circ_{\mathrm{lcc}} = Q^\circ$ if $Q^\circ \cap \mathrm{lcc}_G = \emptyset$ and $Q^\circ_{\mathrm{lcc}} = \{\bot\}$ otherwise.

The set lcc_G^0 of *level-0* limited certain conflict states is the set of blocking states (20). Level $i+1$ adds to this states that can only reach accepting states by passing through a state that can reach a level-i limited certain conflict state using τ-transitions, or using a transition that may lead to a level-i state (21). These sets form an increasing sequence, $\mathrm{lcc}_G^0 \subseteq \mathrm{lcc}_G^1 \subseteq \cdots$, which in the finite-state case converges against the set lcc_G. The abstraction $\textsc{Lcc}(G)$ is constructed by merging these states into a new state $\bot$, and deleting some transitions.

Example 6 Consider again the FSM G in Fig. 4. It holds that $\mathrm{lcc}_G^0 = \{2\}$ and $\mathrm{lcc}_G = \mathrm{lcc}_G^i = \{1,2\}$ for $i \geq 1$. This results in the abstraction $\textsc{Lcc}(G) = H$ (the unreachable state 3 is not shown in the figure).

The limited certain conflicts rule in Def. 11 is described in more detail and is more general than its previous version [7], and therefore a brief proof of its correctness is given here. The argument is based on the following Lemma 14, which describes a crucial property of the set lcc_G of limited certain conflicts: all states in lcc_G are states of certain conflicts, i.e., if such a state is reachable in composition with some test T, then the composition is necessarily blocking.

Lemma 14 Let G and T be two FSMs. If there exists a state pair (x, x_T) such that $G \parallel T \to (x, x_T)$ and $x \in \mathrm{lcc}_G$, then $G \parallel T$ is blocking.

Proof As $x \in \mathrm{lcc}_G$, it holds that $x \in \mathrm{lcc}_G^i$ for some i. The claim is shown by induction on i.

If $i = 0$, then it follows from (20) that $\mathcal{L}^\omega((x, x_T)) \subseteq \mathcal{L}^\omega(x) = \emptyset$, hence $G \parallel T$ with $G \parallel T \to (x, x_T)$ is blocking.

Now assume the claim has been shown for all $x \in \mathrm{lcc}_G^i$, and consider $x \in \mathrm{lcc}_G^{i+1}$. If $\mathcal{L}^\omega((x, x_T)) = \emptyset$ then $G \parallel T$ is blocking, otherwise there exists $t \in \mathcal{L}^\omega((x, x_T)) \subseteq \mathcal{L}^\omega(x)$. By (21), there exists a prefix $u \sqsubseteq t$ such that $x \overset{u}{\Rightarrow} \mathrm{lcc}_G^i$. Then there exists a state pair (y, y_T) of $G \parallel T$ with $y \in \mathrm{lcc}_G^i$ such that $G \parallel T \to (x, x_T) \to (y, y_T)$, i.e., $G \parallel T$ is blocking by inductive assumption. □

Proposition 15 Let G be an FSM. Then $G \simeq_{\text{conf}} \textsc{Lcc}(G)$.

Proof First, to show $\textsc{Lcc}(G) \lesssim_{\text{conf}} G$, let T be an arbitrary FSM such that $\textsc{Lcc}(G) \| T$ is blocking, i.e., $\textsc{Lcc}(G) \parallel T \to (x, x_T)$ with $\mathcal{L}^\omega((x, x_T)) = \emptyset$. It is to be shown that $G \| T$ is blocking. Consider two cases. If $x = \bot$, then it follows by construction (Def. 11) that $G \| T \to \text{lcc}_G \times \{x_T\}$, so $G \| T$ is blocking by Lemma 14. Otherwise, if $x \neq \bot$, then by construction $G \| T \to (x, x_T)$. If $\mathcal{L}^\omega((x, x_T)) = \emptyset$ in $G \| T$ then $G \| T$ is blocking, otherwise there exists a path $(x, x_T) \stackrel{t\omega}{\to}$. As $\textsc{Lcc}(G) \| T$ is blocking, this path cannot exist in $\textsc{Lcc}(G) \parallel T$, i.e., at least one of the transitions on this path exists in G but not in $\textsc{Lcc}(G)$. Then by construction $G \| T \to (x, x_T) \to \text{lcc}_G \times Q_T$, so $G \parallel T$ is blocking by Lemma 14.

Second, to show $G \lesssim_{\text{conf}} \textsc{Lcc}(G)$, let T be an arbitrary FSM such that $\textsc{Lcc}(G) \| T$ is nonblocking. It is to be shown that $G \| T$ is nonblocking, so let $G \| T \to (x, x_T)$. If this path contains a transition present in G but not in $\textsc{Lcc}(G)$ then it follows by construction (Def. 11) that $\textsc{Lcc}(G) \parallel T \to (\bot, x_T)$, which means that $\textsc{Lcc}(G) \parallel T$ is blocking. But $\textsc{Lcc}(G) \| T$ was assumed nonblocking, so the path $G \| T \to (x, x_T)$ also exists in $\textsc{Lcc}(G) \parallel T$. As $\textsc{Lcc}(G) \parallel T$ is nonblocking, there exists a path $(x, x_T) \stackrel{t\omega}{\to}$, which by construction implies $(x, x_T) \stackrel{t\omega}{\to}$ in $G \parallel T$. As $G \parallel T \to (x, x_T)$ was chosen arbitrarily, it follows that $G \parallel T$ is nonblocking. □

By Prop. 15, during compositional nonblocking verification, an FSM G_1 in (9) can be abstracted to get

$$\textsc{Lcc}(G_1) \parallel G_2 \parallel \cdots \parallel G_n \ . \tag{23}$$

An abstract counterexample for (23) is accepted by all the FSMs in the original system (9), up to the point where $\textsc{Lcc}(G_1)$ visits the new state $\bot$. If $\bot$ is encountered, then the path from this point on must be replaced by a path into the limited certain conflicts of the concrete FSM G_1. To ensure that the concrete counterexample reaches a blocking state, it is extended to the lowest level $\text{lcc}^i_{G_1}$ (closest to blocking) possible according to the other FSMs $G_2 \parallel \cdots \parallel G_n$.

Algorithm 2 searches for this extension. Given the abstract counterexample $\tilde{C}$ for (23), it first determines the starting point I for extension. If $\tilde{C}$ starts in $\bot$, then the search starts from the initial states of the concrete FSM G_1 (lines 2–3), otherwise from the last state before $\bot$ in $\tilde{C}$ (lines 5–6). If the abstract counterexample does not contain $\bot$ at all, extension may still be needed as G_1 could reach an accepting state with transitions removed in $\textsc{Lcc}(G_1)$. In this case, the search starts from the end of $\tilde{C}$ (lines 8–9).

Once the set I of start states is determined, the loop on lines 12–15 searches for an extension E from I to the lowest possible level of limited certain conflicts of G_1, which is accepted by the concrete system (9). If no extension can be found, the abstract counterexample is returned unchanged (line 17). Otherwise the result is the extension, possibly preceded by the steps of the abstract counterexample that do not include $\bot$ (line 19 or 21).

The search for the extension on line 12 can be done with a *language inclusion check* [25]. It involves the full composed state space of (9), which may not be feasible to explore in the context of compositional verification. One option is to use the *iterative projection algorithm* [25], which has similar performance characteristics to compositional nonblocking verification.

Algorithm 2: Counterexample Extension after Limited Certain Conflicts

Input: $G_1, \ldots, G_n$ where $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_i^\circ \rangle$
Input: abstract counterexample $\tilde{C} : \tilde{y}^0 \stackrel{\sigma_1}{\rightarrow} \cdots \stackrel{\sigma_k}{\rightarrow} \tilde{y}^k$ for $\textsc{Lcc}(G_1) \parallel G_2 \parallel \cdots \parallel G_n$
where $\tilde{y}^i = (x_1^i, x_2^i, \ldots, x_n^i)$ for $0 \leq i \leq k$
Output: concrete counterexample for $G_1 \parallel \cdots \parallel G_n$

```
if x_1^0 = ⊥ then
    j := 0
    I := (Q_1° ∩ lcc_{G_1}) × {(x_2^0, ..., x_n^0)}
else if x_1^i = ⊥ for some 1 ≤ i ≤ k then
    j := min{ i | x_1^i = ⊥ } − 1
    I := {ỹ^j}
else
    j := k;
    I := {ỹ^k}
end
m := min{ i | lcc^i_{G_1} = lcc_{G_1} }
while m ≥ 0 and I → lcc^m_{G_1} × Q_2 × ··· × Q_n in G_1 ∥ ··· ∥ G_n do
    assign E to be the path I → lcc^m_{G_1} × Q_2 × ··· × Q_n
    m := m − 1
end
if E is unassigned then
    return C̃
else if j = 0 then
    return E
else
    return ỹ^0 -σ_1-> ··· -σ_j-> E
end
```

The complexity of Algorithm 2 is dominated by these language inclusion checks. Their number is bounded by $O(m)$, the maximum level of limited certain conflicts. It can be reduced to $O(\log_2 m)$ using *binary search* [29], or to a single check using a modified language inclusion procedure that takes the levels into account. Even so, extension can result in one additional language inclusion check per successful abstraction step, substantially increasing the overall nonblocking check time.

The remainder of this subsection is devoted to the correctness proof of Algorithm 2, which is based on the following two observations. Firstly, an abstract counterexample for $\textsc{Lcc}(G) \parallel T$ is either a concrete counterexample or it takes the concrete system into a state of limited certain conflicts—if the abstract counterexample ends in $\bot$, it becomes a concrete trace into limited certain conflicts by construction of $\textsc{Lcc}(G)$, otherwise this is the claim of Lemma 16 (i) below. Secondly, once a concrete trace into limit certain conflicts is found, this trace is either already a concrete counterexample, or it can be extended deeper into limited certain conflicts—this is stated in Lemma 16 (ii).

Lemma 16 Let G and T be FSMs.

(i) If $C : (x_G, x_T) \xrightarrow{s} (y_G, y_T)$ with $y_G \neq \bot$ is a counterexample to the nonblocking property of $\textsc{Lcc}(G) \parallel T$, then C is a counterexample to the nonblocking property of $G \parallel T$ or $(y_G, y_T) \to \mathrm{lcc}_G \times Q_T$ in $G \parallel T$.

(ii) If $C : (x_G, x_T) \xrightarrow{s} (y_G, y_T)$ with $y_G \in \mathrm{lcc}_G^i$ is a path in $G \parallel T$, then C is a counterexample to the nonblocking property of $G \| T$ or $(y_G, y_T) \to \mathrm{lcc}_G^{i-1} \times Q_T$ in $G \parallel T$.

Proof (i) If C is a path in $\textsc{Lcc}(G) \parallel T$ that does not end in $\bot$, it follows by construction (Def. 11) that C does not include $\bot$ and is a path in $G \parallel T$. Assume C is not a counterexample to the nonblocking property of $G \parallel T$, i.e., there is a path $(y_G, y_T) \xrightarrow{t\omega}$ in $G \parallel T$. As C is a counterexample to the nonblocking property of $\textsc{Lcc}(G) \parallel T$, some transition on this path has been removed in $\textsc{Lcc}(G)$, i.e., $(y_G, y_T) \to (v_G, v_T) \xrightarrow{\sigma} (w_G, w_T)$ where $v_G \xrightarrow{\sigma} w_G$ but not $v_G \xrightarrow{\sigma}_{\mathrm{lcc}} w_G$. This means by construction that $w_G \in \mathrm{lcc}_G$ or $v_G \overset{P(\sigma)}{\Longrightarrow} \mathrm{lcc}_G$. In both cases, this gives rise to a path $(y_G, y_T) \to \mathrm{lcc}_G \times Q_T$ in $G \parallel T$.

(ii) Assume that the given path C is not a counterexample to the nonblocking property of $G \parallel T$. Then there is a path $(y_G, y_T) \xrightarrow{t\omega}$ in $G \parallel T$, and it is possible to write $y_G = x_0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} x_k \xrightarrow{\omega}$. As $y_G \in \mathrm{lcc}_G^i$, it follows from (20) and (21) that $i > 0$ and there exists $j \geq 0$ such that $j \leq k$ and $x_j \overset{\varepsilon}{\Rightarrow} \mathrm{lcc}_G^{i-1}$, or $j < k$ and $x_j \overset{\sigma_{j+1}}{\Longrightarrow} \mathrm{lcc}_G^{i-1}$. It follows that $x_0 \overset{\sigma_1}{\Rightarrow} \cdots \overset{\sigma_l}{\Rightarrow} \mathrm{lcc}_G^{i-1}$ for some $l \leq k$, which implies $(y_G, y_T) \to \mathrm{lcc}_G^{i-1} \times Q_T$ in $G \parallel T$. □

Given the results from Lemma 16, the correctness of Algorithm 2 is proved by distinguishing the cases resulting from the if-statements in the algorithm.

Proposition 17 Let $G_1, \ldots, G_n$ be FSMs. If $\tilde{C}$ is a counterexample to the nonblocking property of $\textsc{Lcc}(G_1) \parallel G_2 \parallel \cdots \parallel G_n$, then Algorithm 2 terminates and returns a counterexample to the nonblocking property of $G_1 \parallel \cdots \parallel G_n$.

Proof Let $\tilde{C} : \tilde{y}^0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} \tilde{y}^k$, with $\tilde{y}^i = (x_1^i, x_2^i, \ldots, x_n^i)$ for $0 \leq i \leq k$, be a counterexample to the nonblocking property of $\textsc{Lcc}(G_1) \parallel G_2 \parallel \cdots \parallel G_n$. Consider three cases.

Case 1: $x_1^0 = \bot$. Lines 2 and 3 of Algorithm 2 assign $j = 0$ and $I = (Q_1^\circ \cap \mathrm{lcc}_{G_1}) \times \{(x_2^0, \ldots, x_n^0)\}$. As $x_1^0 = \bot$, clearly $\bot \in Q_{\mathrm{lcc}}^\circ$ which by construction (Def. 11) implies $Q_1^\circ \cap \mathrm{lcc}_{G_1} \neq \emptyset$. Then the empty path satisfies $I \to \mathrm{lcc}_{G_1}^m \times Q_2 \times \cdots \times Q_n$ on line 12 and becomes the path E accepted by $G \parallel T$. The loop may perform further iterations, until $m = 0$ or E is a path to $\mathrm{lcc}_{G_1}^{m+1} \times Q_2 \times \cdots \times Q_n$ that cannot be extended to $\mathrm{lcc}_{G_1}^m \times Q_2 \times \cdots \times Q_n$. Then E is a counterexample to the nonblocking property of $G \parallel T$ by Lemma 16 (ii), which given $j = 0$ is returned from line 19.

Case 2: $x_1^i = \bot$, *where* $i \geq 1$ *is the smallest such index.* Lines 5 and 6 assign $j = i - 1$ and $I = \{\tilde{y}^j\}$, where $\tilde{y}^0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_j} \tilde{y}^j$ is a path in $G \parallel T$ and $x_1^j \xrightarrow{\sigma_i} \bot$ in $\textsc{Lcc}(G)$. Then $x_1^j \xrightarrow{\sigma_i} \mathrm{lcc}_G$ by construction (Def. 11). Thus there exists a path E that satisfies $I \to \mathrm{lcc}_{G_1}^m \times Q_2 \times \cdots \times Q_n$ on line 12. The loop may perform further iterations, until $m = 0$ or E is a path to $\mathrm{lcc}_{G_1}^{m+1} \times Q_2 \times \cdots \times Q_n$ that cannot be extended to $\mathrm{lcc}_{G_1}^m \times Q_2 \times \cdots \times Q_n$. Then $\tilde{y}^0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_j} E$ is a counterexample to the nonblocking property of $G \parallel T$ by Lemma 16 (ii), which is returned from line 19 or 21.

Table 1 Experimental results.

Model				No LCC			LCC				
Name	n	State space	CE min	Total [s]	Exp [s]	CE len	Total [s]	Exp [s]	Ext [s]	Ext #	CE len
agvb	17	$2.29 \cdot 10^7$	6	0.2	0.0	18	0.3	0.0	0.1	4	17
aip0alps	35	$3.00 \cdot 10^8$	4	0.3	0.0	19	0.3	0.0	0.1	1	20
aip0tough	60	$1.02 \cdot 10^{10}$	39	7.1	0.1	149	6.9	0.1	6.2	5	39
aip1efa ⟨16⟩	50	$9.50 \cdot 10^{12}$	153	33.9	0.2	185	35.5	0.6	1.3	1	185
aip1efa ⟨24⟩	50	$1.83 \cdot 10^{13}$	153	27.9	0.1	185	27.8	0.1	0.0	0	185
ct17	67	$3.91 \cdot 10^{22}$	3	0.5	0.0	5	0.3	0.0	0.0	1	5
fencaiwon09b	31	$8.93 \cdot 10^7$	225	0.4	0.0	249	0.4	0.0	0.0	1	249
fencaiwon09s	29	$3.00 \cdot 10^8$	2	0.3	0.0	25	0.3	0.0	0.0	1	6
fms2003	30	$1.70 \cdot 10^7$	20	1.3	0.1	22	0.4	0.0	0.0	1	20
ftechnik	36	$1.21 \cdot 10^8$	0	0.4	0.0	3	0.5	0.0	0.1	3	3
prime_sieve4b	16	$1.17 \cdot 10^{20}$	31	5.0	0.4	32	8.1	0.6	3.8	10	32
psl_big	37	$3.87 \cdot 10^7$	13	0.4	0.0	13	0.3	0.0	0.0	1	24
psl_partleft	39	$7.69 \cdot 10^7$	4	5.7	0.1	9	0.7	0.0	0.4	3	15
psl_restart	37	$3.87 \cdot 10^7$	8	1.3	0.0	28	0.9	0.0	0.3	5	26
tbed_ctct	84	$3.94 \cdot 10^{13}$	0	6.7	0.1	0	471.9	0.3	465.8	11	203
tbed_hisc1	184	$2.87 \cdot 10^{17}$	19	2.4	0.1	22	2.8	0.1	1.6	13	59
tbed_noderailb	84	$3.20 \cdot 10^{12}$	2	3.4	0.2	4	47.4	0.2	44.2	8	4
tip3_bad	54	$5.25 \cdot 10^{10}$	16	0.8	0.1	24	0.8	0.1	0.0	0	24
verriegel3b	52	$1.32 \cdot 10^9$	4	0.7	0.0	45	0.8	0.0	0.4	3	4
verriegel4b	64	$6.26 \cdot 10^{10}$	4	0.9	0.1	54	1.0	0.0	0.5	3	4
6linka	53	$2.45 \cdot 10^{14}$	5	0.3	0.0	6	0.3	0.0	0.0	1	6
6linki	53	$2.75 \cdot 10^{14}$	5	0.3	0.0	6	0.3	0.0	0.0	1	6
6linkp	48	$4.43 \cdot 10^{14}$	1	0.4	0.0	11	0.2	0.0	0.0	1	11
6linkre	59	$6.21 \cdot 10^{13}$	90	0.4	0.1	102	0.5	0.0	0.1	11	108

Case 3: $\tilde{C}$ does not include $\perp$. Lines 8 and 9 assign $j = k$ and $I = \{\tilde{y}^j\}$. In this case, $\tilde{C}$ is a path accepted by $G \parallel T$. If the loop-entry condition $I \to \mathrm{lcc}_{G_1}^m \times Q_2 \times \cdots \times Q_n = \mathrm{lcc}_{G_1} \times Q_2 \times \cdots \times Q_n$ on line 12 fails, then $\tilde{C}$ is a counterexample to the nonblocking property of $G \parallel T$ by Lemma 16 (i), which is returned from line 17. Otherwise the proof continues as in case 2 above. □

4 Experimental Results

The counterexample expansion procedure is part of the compositional conflict check in Supremica [1]. It has been used to compute counterexamples for 24 examples. The test suite includes complex industrial models and case studies from different application areas such as manufacturing systems, automotive electronics, and communication protocols [21]. The experiments were run on a standard desktop PC using a single 3.3 GHz microprocessor and 8 GiB of RAM.

Each model was verified with and without limited certain conflicts (LCC). The abstraction sequence consists of τ-loop removal, observation equivalent transition removal, marking removal, the Silent Incoming Rule, the Only Silent Outgoing Rule, the Silent Continuation Rule, the Active Events Rule, possibly the Limited Certain Conflicts Rule, observation equivalence, and marking saturation [7, 21].

Table 1 shows the results of the experiments. It shows for each model, the number of FSMs (n), the number of reachable states in the synchronous composition (State space), and the shortest possible counterexample length (CE min). Then

it shows for each test the combined runtime of verification and counterexample computation (Total), the total time taken by Algorithm 1 (Exp), and the length of the computed counterexample (CE len). Algorithm 2 is only needed with limited certain conflicts, where the table shows the time taken by Algorithm 2 (Ext) and the number of language inclusion checks (Ext #).

The compositional conflict check algorithm proves all these models to be blocking within seconds. While the counterexample expansion times of Algorithm 1 are insignificant, counterexample extension by Algorithm 2 adds substantial runtime to the **tbed_ctct** and **tbed_noderailb** tests, by far outweighing the small verification time benefit from the improved abstraction. In other cases such as **psl_partleft** the overall performance improves with limited certain conflicts. The difference is likely due to the time when limited certain conflicts are triggered: late during the abstraction process the number of FSMs and the extension effort are small.

The computed counterexamples are rarely the shortest possible, although close to the minimum in several cases. While Algorithm 1 guarantees a shortest result, this is not the case for Algorithm 2. Either way, minimality in individual steps does not ensure a shortest result overall. In some cases, e.g., **aip0tough** and **verriegel4b**, limited certain conflicts lead to shorter counterexamples, while in other cases such as **tbed_ctct** the opposite is the case. Although shorter counterexamples are usually preferable, extension into certain conflicts makes the counterexample more specific and can add valuable information.

5 Counterexample Algorithm for the General Case

The previously given methods for counterexample expansion and extension are closely linked to specific abstraction rules. This allows for efficient computation, but it requires specific considerations for every abstraction used in a compositional verification algorithm. As there is an ever-increasing number of conflict-preserving abstraction rules [13, 16, 21, 23, 26, 27], such specific algorithms may not always be readily available.

As an alternative, this section proposes an algorithm to compute a concrete counterexample for $G_1 \| \cdots \| G_n$ from an abstract counterexample for $H_1 \| G_2 \| \cdots \| G_n$ based only on the assumption that the abstraction H_1 is less conflicting than G_1, i.e., $H_1 \lesssim_{\text{conf}} G_1$. Being less conflicting is enough for counterexample computation because it ensures that, if $H_1 \parallel G_2 \parallel \cdots \parallel G_n$ is blocking, then $G_1 \parallel \cdots \parallel G_n$ is also blocking, so a counterexample to the nonblocking property exists in both cases.

The algorithm proposed here is based on the conflict preorder algorithm [28]. Section 5.1 introduces the necessary concepts about this algorithm, and based on these Section 5.2 shows how to compute a concrete counterexample.

5.1 Concepts from Conflict Preorder Algorithm

The conflict preorder algorithm [28] determines for two nondeterministic FSMs G and H whether H is less conflicting than G. To compare two FSMs according to the conflict preorder, it is necessary to identify sets of states the two FSMs may reach under the same input. This is done using the well-known *subset construction* [11].

To capture termination, the usual powerset state space is extended by a special state ω entered only after termination.

Definition 12 [28] The *deterministic state space* of FSM $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ is

$$Q_G^{\det} = 2^Q \cup \{\omega\} \ , \tag{24}$$

and the *deterministic transition function* $\delta_G^{\det} \colon Q_G^{\det} \times \Sigma_\omega \rightarrow Q_G^{\det}$ for G is defined as

$$\delta_G^{\det}(X, \sigma) = \begin{cases} \omega, & \text{if } \sigma = \omega \text{ and } X \stackrel{\omega}{\Rightarrow}; \\ \{\, y \in Q \mid X \stackrel{\sigma}{\Rightarrow} y \,\}, & \text{otherwise.} \end{cases} \tag{25}$$

The deterministic transition function $\delta_G^{\det}$ is extended to traces $s \in \Sigma^* \cup \Sigma^*\omega$ in the standard way. Note that $\delta_G^{\det}(X, s)$ is defined for every trace $s \in \Sigma^* \cup \Sigma^*\omega$; if none of the states in X accept the trace s, this is indicated by $\delta_G^{\det}(X, s) = \emptyset$. This is also true for termination: if ω is enabled at some state in X, then $\delta_G^{\det}(X, \omega) = \omega$, otherwise $\delta_G^{\det}(X, \omega) = \emptyset$.

In order to compare two FSMs G and H with respect to possible conflicts, *pairs* of state sets of the subset construction of G and H are considered. Therefore, the deterministic transition function is also applied to pairs $\mathbf{X} = (X_H, X_G)$ of state sets $X_G \subseteq Q_G$ and $X_H \subseteq Q_H$,

$$\delta_{H,G}^{\det}(\mathbf{X}, s) = \delta_{H,G}^{\det}(X_H, X_G, s) = (\delta_H^{\det}(X_H, s), \delta_G^{\det}(X_G, s)) \ . \tag{26}$$

To determine whether $H \lesssim_{\text{conf}} G$, the algorithm [28] checks all states $x_H \in Q_H$ against matching state sets $X_G \subseteq Q_G$ and determines whether all possible conflicts of x_H are also present in X_G. It cannot always be determined directly whether a state $x_H \in Q_H$ is less conflicting than a state set $X_G \subseteq Q_G$. Generally, it is necessary also to consider the deterministic successors of x_G and X_H. Therefore, the following definition considers pairs (X_H, X_G) of state sets.

Definition 13 [28] Let G and H be FSMs. The set $\mathrm{LC}(H, G) \subseteq Q_H^{\det} \times Q_G^{\det}$ of *less conflicting pairs (LC-pairs)* for H and G is inductively defined by

$$\mathrm{LC}^0(H, G) = (\{\omega\} \times Q_G^{\det}) \cup \{\, (X_H, X_G) \mid X_G \subseteq Q_G \text{ and there exists } x_G \in X_G \text{ such that } \mathcal{L}^\omega(x_G) = \emptyset \,\} \ ; \tag{27}$$

$$\mathrm{LC}^{n+1}(H, G) = \{\, (X_H, X_G) \mid \text{there is a state } x_G \in X_G \text{ such that for all } t \in \mathcal{L}^\omega(x_G) \text{ there exists } r \sqsubseteq t\omega \text{ such that } \delta_{H,G}^{\det}(X_H, X_G, r) \in \mathrm{LC}^i(H, G) \text{ for some } i \leq n \,\} \ ; \tag{28}$$

$$\mathrm{LC}(H, G) = \bigcup_{n \geq 0} \mathrm{LC}^n(H, G) \ . \tag{29}$$

Note that $X_G \neq \emptyset$ and $X_G \neq \omega$ for every LC-pair $(X_H, X_G) \in \mathrm{LC}(H, G)$ with $X_H \neq \omega$. The idea of Def. 13 is to classify a pair (X_H, X_G) as less conflicting, if every test FSM T that is nonconflicting in combination with each of the states in X_H can terminate with at least one trace from the accepting language of X_G. This leads to the following main result for testing the conflict preorder.

Theorem 18 [28] Let $G = \langle \Sigma_G, Q_G, \rightarrow_G, Q_G^\circ \rangle$ and $H = \langle \Sigma_H, Q_H, \rightarrow_H, Q_H^\circ \rangle$ be two FSMs. Then $H \lesssim_{\text{conf}} G$ if and only if for all $s \in \textsc{NConf}(G)$ and all $x_H \in Q_H$ such that $H \stackrel{s}{\Rightarrow} x_H$ it holds that $(\{x_H\}, \delta_G^{\det}(Q_G^\circ, s)) \in \mathrm{LC}(H, G)$.

Given the less conflicting pairs for two FSMs G and H, it is possible to determine whether $H \lesssim_{\text{conf}} G$. This is the case if all the pairs $(\{x_H\}, X_G)$ with $H \stackrel{s}{\Rightarrow} x_H$ and $\delta_G^{\text{det}}(Q_G^\circ, s) = X_G$ for some $s \in \text{NCONF}(G)$ are LC-pairs. The condition does not apply to traces of certain conflicts, because traces $s \in \text{CONF}(G)$ are not necessarily in the language $\mathcal{L}(G)$ of G. Therefore, checking whether $H \lesssim_{\text{conf}} G$ in general also requires computation of the set of certain conflicts of G, which can be done with a direct algorithm [14] or using the following characterisation based on LC-pairs.

Theorem 19 [28] Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an FSM. Then

$$\text{CONF}(G) = \{\, s \in \Sigma^* \mid (\emptyset, \delta_G^{\text{det}}(Q^\circ, r)) \in \text{LC}(H, G) \text{ for some } r \sqsubseteq s \,\} \;, \tag{30}$$

for any FSM H.

This result can be explained by the observation that, if $(\emptyset, X_G)$ is an LC-pair then, since termination is impossible from $\emptyset$, conflict must also be present in X_G. In this case, every trace leading to X_G must be a trace of certain conflicts, and so must be all its extensions. The characterisation of certain conflicts using LC-pairs helps to integrate the computation of certain conflicts with the rest of the conflict preorder algorithm, and likewise with counterexample expansion and extension.

While the Theorem 18 shows that LC-pairs are crucial for checking the conflict preorder, it is not immediately clear how to determine whether a given state-set pair is an LC-pair. The computation of all LC-pairs for two FSMs H and G is done in a nested iteration. The set $\text{LC}^0(H, G)$ is given by (27), and assuming that the set $\text{LC}^n(H, G)$ is already known, the set $\text{LC}^{n+1}(H, G)$ is computed in a secondary iteration based on *more conflicting triples*.

Definition 14 [28] Let $G = \langle \Sigma_G, Q_G, \to_G, Q_G^\circ \rangle$ and $H = \langle \Sigma_H, Q_H, \to_H, Q_H^\circ \rangle$ be two FSMs. The set $\text{MC}^n(H, G) \subseteq Q_H^{\text{det}} \times Q_G^{\text{det}} \times Q_G$ of n^{th}-level *more conflicting triples (MC-triples)* for H and G is defined inductively as follows.

$$\text{MC}_0^n(H, G) = \{\, (\emptyset, \omega, x_G) \mid x_G \in Q_G \,\} \;; \tag{31}$$

$$\begin{aligned} \text{MC}_{m+1}^n(H, G) = \{\, (X_H, X_G, x_G) \mid\ & (X_H, X_G) \notin \text{LC}^n(H, G) \text{ and } x_G \in X_G \\ & \text{and there exists } (Y_H, Y_G, y_G) \in \text{MC}_m^n(H, G) \text{ and } \sigma \in \Sigma_\omega \\ & \text{such that } \delta_{H,G}^{\text{det}}(X_H, X_G, \sigma) = (Y_H, Y_G) \text{ and } x_G \stackrel{\sigma}{\Rightarrow} y_G \,\} \;; \end{aligned} \tag{32}$$

$$\text{MC}^n(H, G) = \bigcup_{m \geq 0} \text{MC}_m^n(H, G) \;. \tag{33}$$

For (X_H, X_G) to be an LC-pair, according to Def. 13 there must be a state $x_G \in X_G$ such that every trace that takes x_G to termination in G has a prefix that leads to another less conflicting pair. A triple (X_H, X_G, x_G) is considered "more conflicting" if (X_H, X_G) is not yet known to be a less conflicting pair, and the state $x_G \in X_G$ cannot be used to confirm the above property. This is ensured by (32), which stipulates the existence of a successor MC-triple of lower level, from where termination in G is again possible without visiting $\text{LC}^n(H, G)$. Then, if all the triples (X_H, X_G, x_G) with $x_G \in X_G$ are MC-triples, it follows that (X_H, X_G) cannot be an LC-pair. This observation leads to the following characterisation of LC-pairs based on MC-triples.

Theorem 20 [28] Let G and H be two FSMs, and let $n \in \mathbb{N}_0$. Then

$$\begin{aligned} \mathrm{LC}^{n+1}(H,G) = \{\, (X_H, X_G) \in Q_H^{\det} \times Q_G^{\det} \mid \\ (X_H, X_G, x_G) \notin \mathrm{MC}^n(H,G) \text{ for some } x_G \in X_G \,\} \,. \end{aligned} \tag{34}$$

MC-triples can be computed directly based on Def. 14, so that this result allows for the effective computation of LC-pairs, and an algorithm to test the conflict preorder can be described. It involves computing the relevant state-set pairs according to Theorem 18 and all their successors, as well as the induced MC-triples along with their successors, until it can be determined whether all the relevant pairs are LC-pairs [28].

5.2 Computation of Concrete Counterexample using LC-pairs

Assuming that H_1 is a less conflicting abstraction of G_1, this section shows how the LC-pairs $\mathrm{LC}(H_1, G_1)$ are used to compute a concrete counterexample for $G_1 \| T$ from an abstract counterexample for $H_1 \,\|\, T$, where $T = G_2 \,\|\, \cdots \,\|\, G_n$ represents the unchanged part of the system. If an abstract counterexample exists for $H_1 \,\|\, T$,

$$\tilde{C} : (x_H^\circ, x_T^\circ) \overset{s}{\Rightarrow} (x_H, x_T) \,, \tag{35}$$

then its end state must be blocking, $\mathcal{L}^\omega(x_H) \cap \mathcal{L}^\omega(x_T) = \emptyset$. Moreover, if the trace s is also accepted by the original FSM G and does not belong to certain conflicts, then by Theorem 18 it holds that $(\{x_H\}, \delta_G^{\det}(Q_G^\circ, s))$ is an LC-pair. Such an LC-pair is called *critical* for x_T as it plays a crucial role for counterexample expansion and extension.

Definition 15 Let G, H, and T be FSMs, and let x_T be a state of T. An LC-pair $(X_H, X_G) \in \mathrm{LC}(H,G)$ is called *critical for* x_T if $X_H \neq \omega$ and $\mathcal{L}^\omega(X_H) \cap \mathcal{L}^\omega(x_T) = \emptyset$.

An LC-pair (X_H, X_G) is critical for x_T, if all the states recorded in X_H for the abstract FSM are blocking in combination with x_T. A critical LC-pair also must not already be terminated, $X_H \neq \omega$, as this cannot arise in a counterexample to the nonblocking property. Usually, an abstract counterexample such as (35) gives rise to a critical LC-pair according to Theorem 18, but there is a special case in combination with certain conflicts. If $s \in \textsc{Conf}(G)$, then s is not necessarily accepted by the original FSM G, and in this case the trace needs to be reduced to a prefix. The following result holds.

Proposition 21 Let G and H be FSMs such that $H \lesssim_{\mathrm{conf}} G$, and let T be a third FSM such that C: $(x_H^\circ, x_T^\circ) \overset{s}{\Rightarrow} (x_H, x_T)$ is a counterexample to the nonblocking property of $H \,\|\, T$. Then there exists a critical LC-pair for x_T in $\mathrm{LC}(H,G)$

Proof Write $G = \langle \Sigma_G, Q_G, \rightarrow_G, Q_G^\circ \rangle$, and consider two cases depending on whether the trace s is among the certain conflicts of G or not.

First consider the case that $s \in \textsc{NConf}(G)$. Then, noting that $H \lesssim_{\mathrm{conf}} G$ and $H \overset{s}{\Rightarrow} x_H$, it follows from Theorem 18 that $(\{x_H\}, X_G) \in \mathrm{LC}(H,G)$ where $\delta_G^{\det}(Q_G^\circ, s) = X_G$. Let $X_H = \{x_H\}$. As C is a counterexample to the nonblocking property of $H \,\|\, T$, it is clear that $\mathcal{L}^\omega(X_H) \cap \mathcal{L}^\omega(x_T) = \mathcal{L}^\omega(x_H) \cap \mathcal{L}^\omega(x_T) = \emptyset$.

Second consider the case $s \in \textsc{Conf}(G)$. Then by Theorem 19 there exists a prefix $r \sqsubseteq s$ such that $(\emptyset, X_G) \in \mathrm{LC}(H, G)$ where $\delta_G^{\det}(Q_G^\circ, r) = X_G$. With $X_H = \emptyset$, it is clear that $\mathcal{L}^\omega(X_H) \cap \mathcal{L}^\omega(x_T) = \emptyset$.

In both cases, this gives an LC-pair $(X_H, X_G) \in \mathrm{LC}(H, G)$ with $X_H \neq \omega$ and $\mathcal{L}^\omega(X_H) \cap \mathcal{L}^\omega(x_T) = \emptyset$, i.e., critical for x_T. □

Thus, a critical LC-pair exists for every concrete counterexample and less conflicting abstraction, and the above proof suggests how it can be found depending on whether the counterexample trace belongs to certain conflicts or not.

A critical LC-pair ensures that the abstract FSM H is blocking in combination with the state x_T of T, but this is not yet guaranteed for the original FSM G. To find a state of G blocking with x_T, a process of extension similar to Algorithm 2 is necessary. This process is based on the following result.

Proposition 22 Let G, H, and T be FSMs, and let x_T be a state of T. Let $(X_H, X_G) \in \mathrm{LC}^{n+1}(H, G)$ be an LC-pair that is critical for x_T. Then at least one of the following conditions holds.

(i) There exists a trace $t \in \Sigma^*$ such that $x_T \overset{t}{\Rightarrow} y_T$ and $\delta_{H,G}^{\det}(X_H, X_G, t) \in \mathrm{LC}^n(H, G)$ is critical for y_T.
(ii) For every state $x_G \in X_G$ such that $(X_H, X_G, x_G) \notin \mathrm{MC}^n(H, G)$ it holds that $\mathcal{L}^\omega(x_G) \cap \mathcal{L}^\omega(x_T) = \emptyset$.

Proof Consider two cases.

Case 1: there exists a trace $t \in \Sigma^*$ such that $\delta_{H,G}^{\det}(X_H, X_G, t) \in \mathrm{LC}^n(H, G)$ and $x_T \overset{t}{\Rightarrow} y_T$. Let $\delta_{H,G}^{\det}(X_H, X_G, t) = (Y_H, Y_G)$. Then $Y_H \neq \omega$ as $t \in \Sigma^*$, and $\mathcal{L}^\omega(Y_H) \cap \mathcal{L}^\omega(y_T) = \emptyset$ as (X_H, X_G) is critical for x_T and thus $\mathcal{L}^\omega(X_H) \cap \mathcal{L}^\omega(x_T) = \emptyset$. Therefore, (Y_H, Y_G) is critical for y_T and condition (i) holds.

Case 2: there does not exist any trace $t \in \Sigma^*$ with $\delta_{H,G}^{\det}(X_H, X_G, t) \in \mathrm{LC}^n(H, G)$ and $x_T \overset{t}{\Rightarrow} y_T$. It will be shown by contradiction that condition (ii) holds. So let $x_G \in X_G$ such that $(X_H, X_G, x_G) \notin \mathrm{MC}^n(H, G)$, and assume $\mathcal{L}^\omega(x_G) \cap \mathcal{L}^\omega(x_T) \neq \emptyset$. Then there exists $u \in \mathcal{L}^\omega(x_G) \cap \mathcal{L}^\omega(x_T)$. Note that $u \notin \mathcal{L}^\omega(X_H)$ because $\mathcal{L}^\omega(X_H) \cap \mathcal{L}^\omega(x_T) = \emptyset$ for the critical LC-pair (X_H, X_G). Let $s = u\omega$. From $u \in \mathcal{L}^\omega(x_G)$ it follows that $x_G \overset{s}{\Rightarrow}$ and $\delta_{H,G}^{\det}(X_H, X_G, s) = (\emptyset, \omega)$. Write $s = \sigma_m \cdots \sigma_0$ (where $\sigma_0 = \omega$) and $\delta_{H,G}^{\det}(X_H, X_G, \sigma_m \cdots \sigma_i) = (X_H^i, X_G^i)$ for $i = 0, \ldots, m+1$, and let the path $x_G \overset{s}{\Rightarrow}$ be

$$x_G = x_G^{m+1} \overset{\sigma_m}{\Longrightarrow} x_G^m \overset{\sigma_{m-1}}{\Longrightarrow} \cdots \overset{\sigma_1}{\Rightarrow} x_G^1 \overset{\sigma_0}{\Rightarrow} x_G^0 \ . \tag{36}$$

Note that $x_G^i \in X_G^i$ for $i = 1, \ldots, m+1$. It will be shown by induction on i that $(X_H^i, X_G^i, x_G^i) \in \mathrm{MC}_i^n(H, G)$ for $i = 0, \ldots, m+1$.

In the base case, $i = 0$, it holds that $(X_H^0, X_G^0) = \delta_{H,G}^{\det}(X_H, X_G, s) = (\emptyset, \omega)$ and thus $(X_H, X_G, x_G) = (\emptyset, \omega, x_G) \in \mathrm{MC}_0^n(H, G)$ by (31)

Now consider $(X_H^{i+1}, X_G^{i+1}, x_G^{i+1})$. Then $\delta_{H,G}^{\det}(X_H^{i+1}, X_G^{i+1}, \sigma_i) = (X_H^i, X_G^i)$ and $x_G^{i+1} \overset{\sigma_i}{\Rightarrow} x_G^i$, and $(X_H^i, X_G^i, x_G^i) \in \mathrm{MC}_i^n(H, G)$ holds by inductive assumption. Also $\sigma_m \cdots \sigma_{i+1} \in \Sigma^*$ (only $\sigma_0 = \omega$) and $\sigma_m \cdots \sigma_{i+1} \sqsubseteq s \in \mathcal{L}(x_T)$, and therefore $(X_H^{i+1}, X_G^{i+1}) = \delta_{H,G}^{\det}(X_H, X_G, \sigma_m \cdots \sigma_{i+1}) \notin \mathrm{LC}^n(H, G)$ by the assumption of this case. It follows that $(X_H^{i+1}, X_G^{i+1}, x_G^{i+1}) \in \mathrm{MC}_{i+1}^n(H, G)$ by (32).

This completes the induction and shows that $(X_H, X_G, x_G) = (X_H^{m+1}, X_G^{m+1}, x_G^{m+1}) \in \mathrm{MC}_{m+1}^n(H, G) \subseteq \mathrm{MC}^n(H, G)$. But this contradicts the assumption that $(X_H, X_G, x_G) \notin \mathrm{MC}^n(H, G)$. □

Algorithm 3: Counterexample Expansion and Extension with LC-pairs

Input: $G_1, \ldots, G_n$ where $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_i^\circ \rangle$, and $H_1 \lesssim_{\text{conf}} G_1$
Input: abstract counterexample $\tilde{C} : \tilde{y}^0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} \tilde{y}^k$ for $H_1 \parallel G_2 \parallel \cdots \parallel G_n$, where $\tilde{y}^i = (\tilde{x}_1^i, x_2^i, \ldots, x_n^i)$ for $0 \leq i \leq k$
Output: concrete counterexample C for $G_1 \parallel \cdots \parallel G_n$

1 compute all LC-pairs for H_1 and G_1 reachable from $(\emptyset, Q_1^\circ)$
2 **if** $(\emptyset, \delta_1^{\text{det}}(Q_1^\circ, r)) \in \text{LC}(H_1, G_1)$ for some $r \sqsubseteq P(\sigma_1 \cdots \sigma_k)$ **then**
3 | let s be the shortest such trace r
4 | $\mathbf{X} := (\emptyset, \delta_{G_1}^{\text{det}}(Q_1^\circ, s))$
5 **else**
6 | $s := P(\sigma_1 \cdots \sigma_k)$
7 | $\mathbf{X} := (\{\tilde{x}_1^k\}, \delta_{G_1}^{\text{det}}(Q_1^\circ, s))$
8 | compute all LC-pairs for H_1 and G_1 reachable from $\mathbf{X}$
9 **end**
10 $m := \min\{\, l \geq 0 \mid \mathbf{X} \in \text{LC}^l(H_1, G_1) \,\}$
11 construct $C_T : (x_2^0, \ldots, x_n^0) \overset{s}{\Rightarrow} (x_2^j, \ldots, x_n^j)$ from C
12 $x_T := (x_2^j, \ldots, x_n^j)$
13 **while** $\delta_{H_1,G_1}^{\text{det}}(\mathbf{X}, t) \in \text{LC}^{m-1}(H_1, G_1)$ and $x_T \overset{t}{\Rightarrow} y_T$ for some $t \in \Sigma^*$ **do**
14 | $\mathbf{X} := \delta_{H_1,G_1}^{\text{det}}(\mathbf{X}, t)$
15 | $s := st$
16 | $C_T := C_T \overset{t}{\Rightarrow} y_T$
17 | $x_T := y_T$
18 | $m := \min\{\, l \geq 0 \mid \mathbf{X} \in \text{LC}^l(H_1, G_1) \,\}$
19 **end**
20 let $(X_H, X_G) = \mathbf{X}$
21 choose $x_1 \in X_G$ such that $(X_H, X_G, x_1) \notin \text{MC}^{m-1}(H_1, G_1)$
22 find a path $C_1 : Q_1^\circ \overset{s}{\Rightarrow} x_1$ in G_1
23 construct C by combining C_1 and C_T
24 **return** C

Prop. 22 ensures for a critical LC-pair that, either condition (i) holds and its trace can be extended to reach another critical LC-pair of a lower level, or condition (ii) holds and it contains states of G that are blocking with the state x_T of T. In the latter case, as (X_H, X_G) is an LC-pair, by Theorem 20 there exists a state $x_G \in X_G$ such that (X_H, X_G, x_G) does not form an MC-triple, and such states are blocking with x_T.

Algorithm 3 combines these results to compute a concrete counterexample from an abstract counterexample (35) based on an abstraction step $H_1 \lesssim_{\text{conf}} G_1$. First, lines 1–7 find an initial critical LC-pair $\mathbf{X}$ according to Prop. 21. Line 1 computes the LC-pairs needed for the set of certain conflicts of G_1 according to Theorem 19. Accordingly, if the trace of the abstract counterexample is a trace of certain conflicts, then lines 3–4 reduce the counterexample to its shortest prefix that results in a critical LC-pair $\mathbf{X}$. Otherwise the initial critical LC-pair $\mathbf{X}$ is

constructed from the end state of the abstract counterexample. Both cases follow the construction in the proof of Prop. 21.

Once a critical LC-pair $\mathbf{X}$ has been identified, all its successor LC-pairs need to be explored for the following search for an extension. If the abstract counterexample was in certain conflicts, these LC-pairs have already been computed on line 1, otherwise it is done on line 8 using the already computed pairs. The computation of LC-pairs on lines 1 and 8 is done according to the algorithm [28]. The computation can be stopped as soon as $\mathbf{X}$ is found to be an LC-pair, because its level is known at that point and higher-level LC-pairs are not needed for the following counterexample extension. After the computation, it is necessary to remember the level of each LC-pair, and the triple missing from $\mathrm{MC}^{l-1}(H_1, G_1)$ that led to its inclusion in $\mathrm{LC}^{l}(H_1, G_1)$ according to Theorem 20.

With all relevant LC-pairs at hand, lines 10–19 extend the counterexample according to Prop. 22. During this process, m is the level of the critical LC-pair $\mathbf{X}$, C_T is the projection of the counterexample to the unchanged FSMs $T = G_2 \parallel \cdots \parallel G_n$, and x_T is the last state of C_T. The construction of C_T on line 11 involves the removal of the state components for the abstracted FSM H_1 and any τ-transitions associated with H_1 from C. The loop on lines 13–19 continues as long as condition (i) of Prop. 22 holds. While this is true, there exists a critical LC-pair of a lower level reached by a trace also possible in T, in which case that LC-pair becomes the new critical LC-pair $\mathbf{X}$ and the counterexample is extended accordingly.

The loop must terminate, since the level of $\mathbf{X}$ is decreased with each iteration, and a level of 0 is not possible as this requires a terminated trace according to (27). On termination, condition (i) of Prop. 22 is no longer true, so that condition (ii) must hold. As $\mathbf{X} = (X_H, X_G)$ is an LC-pair, by Theorem 20 there exists a state $x_1 \in X_G$ of the original FSM G_1 such that (X_H, X_G, x_1) is not an MC-triple, and then (x_1, x_T) is a blocking state according to condition (ii). Now it only remains to find a path in G_1 to this state x_1, which must exist by construction as $X_G = \delta_{G_1}^{\mathrm{det}}(Q_1^\circ, s)$, and to build the final concrete counterexample C by adding the state information and τ-transitions for G_1 to C_T. These operations, shown on lines 22 and 23, can be implemented by a search similar to Algorithm 1.

The complexity of Algorithm 3 is dominated by the construction of LC-pairs on lines 1 and 8. Although care is taken to explore only the part of the search space needed for the counterexample, the worst case is the exploration of all LC-pairs, of which there can be $O(2^{g+h})$, where $g = |Q_{G_1}|$ and $h = |Q_{H_1}|$ are the numbers of states of the concrete and abstract FSM respectively. The worst-case time complexity to construct these pairs is $O(|\Sigma| \cdot g^2 \cdot 2^{2g+2h})$ [28].

Another time-consuming operation appears in the loop entry condition on line 13, where the graph of LC-pairs must be searched in composition with the unchanged FSMs T. The inclusion of T is necessary for the same reasons as explained in Section 3.5. As mentioned there, the number of iterations can be reduced and the FSMs, including the graph of LC-pairs, can be abstracted using the iterative projection algorithm [25], but the effort is still huge.

The appeal of Algorithm 3 lies in the fact that it is only based on a less conflicting abstraction and makes no assumptions about specific abstraction rules. It works for all conflict-preserving abstractions, and can even be used if several abstraction rules have been applied in sequence. Unfortunately, its exponential complexity means that it may be impracticable for large discrete event systems.

6 Conclusions

Three algorithms for counterexample computation during compositional nonblocking verification are proposed. The counterexample computation process depends on the type of abstraction used during verification, and in many common cases [7] can be done by counterexample *expansion* or *extension*. Counterexample expansion is fast and simple and often applicable. In other cases, particularly after abstraction by certain conflicts, the more time-consuming algorithm of counterexample extension must be used. It is likely that other abstraction rules not covered in this paper, e.g. [13,16,21,23,26,27], can also be treated by these algorithms, but a closer examination is needed in each case. As an alternative, this paper also proposes a slow but general counterexample algorithm that works for every conflict-preserving abstraction.

References

1. Åkesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: 8th Int. Workshop on Discrete Event Systems, WODES '06, pp. 384–385. IEEE (2006). DOI 10.1109/WODES.2006.382401
2. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and Software Verification. Springer (2001)
3. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems, 2 edn. Springer Science & Business Media, New York, NY, USA (2008)
4. Clarke, E.M., Long, D.E., McMillan, K.L.: Compositional model checking. In: 4th Annual Symp. Logic in Computer Science, pp. 353–362 (1989). DOI 10.1109/LICS.1989.39190
5. Dams, D., Grumberg, O., Gerth, R.: Abstract interpretation of reactive systems: Abstractions preserving ∀CTL*, ∃CTL* and CTL*. In: E.R. Olderog (ed.) IFIP WG2.1/WG2.2/WG2.3 Working Conf. Programming Concepts, Methods and Calculi (PROCOMET), IFIP Transactions, pp. 573–592. Elsevier (1994)
6. De Nicola, R., Hennessy, M.C.B.: Testing equivalences for processes. Theoretical Comput. Sci. **34**(1–2), 83–133 (1984). DOI 10.1016/0304-3975(84)90113-0
7. Flordal, H., Malik, R.: Compositional verification in supervisory control. SIAM J. Control Optim. **48**(3), 1914–1938 (2009). DOI 10.1137/070695526
8. Graf, S., Steffen, B.: Compositional minimization of finite state systems. In: 1990 Workshop on Computer-Aided Verification, *LNCS*, vol. 531, pp. 186–196. Springer (1990). DOI 10.1007/BFb0023732
9. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968). DOI 10.1109/TSSC.1968.300136
10. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
11. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Boston, MA, USA (2001)
12. Huth, M., Ryan, M.: Logic in Computer Science. Cambridge University Press, Cambridge, UK (2004)
13. Lindsey, J.: The set of certain conflicts. Honours project report, Dept. of Computer Science, University of Waikato (2012)
14. Malik, R.: The language of certain conflicts of a nondeterministic process. Working Paper 05/2010, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand (2010). URL `http://hdl.handle.net/10289/4108`
15. Malik, R.: Programming a fast explicit conflict checker. In: 13th Int. Workshop on Discrete Event Systems, WODES '16, pp. 464–469. IEEE (2016). DOI 10.1109/WODES.2016.7497885
16. Malik, R., Leduc, R.: Compositional nonblocking verification using generalised nonblocking abstractions. IEEE Trans. Autom. Control **58**(8), 1–13 (2013). DOI 10.1109/TAC.2013.2248255

17. Malik, R., Streader, D., Reeves, S.: Conflicts and fair testing. Int. J. Found. Comput. Sci. **17**(4), 797–813 (2006). DOI 10.1142/S012905410600411X
18. Malik, R., Ware, S.: Counterexample computation in compositional nonblocking verification. IFAC PapersOnLine **51**(7), 230–235 (2018). DOI 10.1016/j.ifacol.2018.06.334
19. Milner, R.: Communication and concurrency. Series in Computer Science. Prentice-Hall (1989)
20. Pena, P.N., Cury, J.E.R., Lafortune, S.: Verification of nonconflict of supervisors using abstractions. IEEE Trans. Autom. Control **54**(12), 2803–2815 (2009). DOI 10.1109/TAC.2009.2031730
21. Pilbrow, C., Malik, R.: An algorithm for compositional nonblocking verification using special events. Sci. Comput. Programming **113**(2), 119–148 (2015). DOI 10.1016/j.scico.2015.05.010
22. Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proc. IEEE **77**(1), 81–98 (1989). DOI 10.1109/5.21072
23. Su, R., van Schuppen, J.H., Rooda, J.E., Hofkamp, A.T.: Nonconflict check by using sequential automaton abstractions based on weak observation equivalence. Automatica **46**(6), 968–978 (2010). DOI 10.1016/j.automatica.2010.02.025
24. Valmari, A.: Compositionality in state space verification methods. In: 18th Int. Conf. Application and Theory of Petri Nets, *LNCS*, vol. 1091, pp. 29–56. Springer (1996). DOI 10.1007/3-540-61363-3_3
25. Ware, S., Malik, R.: The use of language projection for compositional verification of discrete event systems. In: 9th Int. Workshop on Discrete Event Systems, WODES '08, pp. 322–327. IEEE (2008). DOI 10.1109/WODES.2008.4605966
26. Ware, S., Malik, R.: Conflict-preserving abstraction of discrete event systems using annotated automata. Discrete Event Dyn. Syst. **22**(4), 451–477 (2012). DOI 10.1007/s10626-012-0133-3
27. Ware, S., Malik, R.: Compositional verification of the generalized nonblocking property using abstraction and canonical automata. Int. J. Found. Comput. Sci. **24**(8), 1183–1208 (2013). DOI 10.1142/S0129054113500287
28. Ware, S., Malik, R.: An algorithm to test the conflict preorder. Sci. Comput. Programming **89**(A), 23–40 (2014). DOI 10.1016/j.scico.2013.09.006
29. Wirth, N.: Algorithms and Data Structures. Prentice-Hall (1986)