Counterexample Computation in Compositional Nonblocking Verification

Robi Malik * **Simon Ware** **

* *Department of Computer Science, University of Waikato, Hamilton, New Zealand (e-mail: robi@waikato.ac.nz)*
** *School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore (e-mail: sware@ntu.edu.sg)*

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 a solution, and experimental results show that counterexamples can be computed successfully for several industrial-scale systems.

Keywords: Verification, validation, test; Supervisory control theory; Tools.

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 (Ramadge and Wonham, 1989). 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 (Baier and Katoen, 2008).

Compositional verification (Graf and Steffen, 1990) is an effective alternative that can be used independently of or in combination with symbolic methods. Compositional verification works by *abstracting* or *simplifying* individual components of a large system, gradually reducing the state space and allowing larger systems to be verified in the end. When applied to the nonblocking property, compositional verification requires specific abstraction methods (Flordal and Malik, 2009), and a suitable theory is that of *conflict equivalence* (Malik et al., 2006). Various abstraction rules preserving conflict equivalence have been proposed (Pena et al., 2009; Su et al., 2010; Ware and Malik, 2012; Lindsey, 2012; Pilbrow and Malik, 2015).

If a system fails the nonblocking check, it is 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 livelock or deadlock situation, and it is routinely computed by standard model checking algorithms (Malik, 2016). 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 is addressed by Ware and Malik (2008) for safety properties, but only partly by Lindsey (2012) for the nonblocking property. This paper gives a more complete account of the issues and solutions regarding counterexample computation in compositional nonblocking verification.

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 some experimental results. Lastly, Section 5 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. Their basic building blocks are *events*, which are taken from a finite *alphabet* Σ. The *silent event* $\tau \notin \Sigma$ labels transitions that are only taken by the component under consideration.

Σ^* denotes the set of all finite *traces* of the form $\sigma_1\sigma_2\cdots\sigma_n$ of events from Σ, including the *empty trace* ε. The *concatenation* of two traces $s, t \in \Sigma^*$ is written as st. A subset $L \subseteq \Sigma^*$ is called a *language*. The *standard projection* $P\colon (\Sigma \cup \{\tau\})^* \to \Sigma^*$ is the operation that deletes all silent (τ) events from traces.

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

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$ and extended to traces $s \in (\Sigma \cup \{\tau\})^*$ 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 $s \in (\Sigma \cup \{\tau\})^*$, 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 $\sigma \notin \Sigma \cup \{\tau\}$.

To support silent events, another transition relation $\Rightarrow \subseteq Q \times \Sigma^* \times Q$ is introduced, where $x \stackrel{s}{\Rightarrow} y$ denotes the existence of a trace $t \in (\Sigma \cup \{\tau\})^*$ 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 $\rightarrow$. The *accepting language* of a state or state set x is $\mathcal{L}^\omega(x) = \{\, s \in \Sigma^* \mid x \stackrel{s}{\Rightarrow} Q^\omega \,\}$. For an FSM G, the notation $G \stackrel{s}{\Rightarrow} x$ means $Q^\circ \stackrel{s}{\Rightarrow} x$.

FSMs are synchronised in lock-step (Hoare, 1985). The *synchronous composition* of two FSMs $G = \langle \Sigma_G, Q_G, \rightarrow_G, Q_G^\circ, Q_G^\omega \rangle$ and $H = \langle \Sigma_H, Q_H, \rightarrow_H, Q_H^\circ, Q_H^\omega \rangle$ is

$$G \parallel H \;=\; \langle \Sigma_G \cup \Sigma_H, Q_G \times Q_H, \rightarrow, Q_G^\circ \times Q_H^\circ, Q_G^\omega \times Q_H^\omega \rangle$$

where

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

2.2 The Nonblocking Property

The key liveness property in supervisory control theory is the *nonblocking* property (Ramadge and Wonham, 1989). An FSM is nonblocking if, from every reachable state, an accepting state can be reached.

Definition 2. (Malik et al., 2006) An FSM $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ 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 $x \stackrel{t}{\Rightarrow} Q^\omega$; otherwise G is *blocking*.

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*.

Definition 3. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an FSM. A *counterexample to the nonblocking property* of G is a path

$$x_0 \xrightarrow{\sigma_1} x_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} x_n \tag{1}$$

such that $x_0 \in Q^\circ$, and $\mathcal{L}^\omega(x_n) = \emptyset$.

A counterexample highlights the cause of blocking by showing a path that leads to a faulty (blocking) state. It is an alternating sequence of states and events, starting from the initial state and following transitions of the FSM. Its end state x_n has an empty accepting language, $\mathcal{L}^\omega(x_n) = \emptyset$, or equivalently $x_n \stackrel{t}{\Rightarrow} Q^\omega$ 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.

2.3 Compositional Verification

To reason about the nonblocking property in a compositional way, the notion of *conflict equivalence* is used (Malik et al., 2006). According to process-algebraic testing theory, two FSMs are considered as equivalent if they both respond in the same way to tests (De Nicola and Hennessy, 1984). 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 4. (Malik et al., 2006) Let G and H be two FSMs. G is *less conflicting* than H, written $G \lesssim_{\text{conf}} H$, if for any FSM T such that $G \| T$ is nonblocking, it also holds that $H \| T$ is nonblocking. G and H are *conflict equivalent*, $G \simeq_{\text{conf}} H$, if $G \lesssim_{\text{conf}} H$ and $H \lesssim_{\text{conf}} G$.

When verifying whether a composed system of FSMs

$$G_1 \parallel G_2 \parallel \cdots \parallel G_n\ , \tag{2}$$

is nonblocking, *compositional* methods (Graf and Steffen, 1990; Flordal and Malik, 2009) 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 G, which is the compositional abstraction of (2). The *congruence* properties (Malik et al., 2006) of conflict equivalence ensure that G is nonblocking if and only if the original system (2) is. G typically has fewer states than (2), making it possible to check whether it is nonblocking even though the full composition (2) may be too large to fit in memory.

3. COUNTEREXAMPLE EXPANSION

Assume that a system (2) is found to be blocking at the end of compositional verification, i.e., the final compositional abstraction is blocking. Then standard state exploration algorithms (Malik, 2016) 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 (2).

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 (2) 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 principle using simple abstraction steps, and afterwards Sections 3.2 and 3.3 present expansion algorithms for two typical conflict-preserving abstraction rules.

3.1 Synchronous Composition and Hiding

The simplest abstraction step is that of *synchronous composition*. The system (2) can be replaced by

$$(G_1 \parallel G_2) \parallel G_3 \parallel \cdots \parallel G_n\ . \tag{3}$$

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 (3) 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 (2).

Another simple type of abstraction is *hiding*. Assume that in (2) the events in some $\Upsilon \subseteq \Sigma$ appear only in G_1 and in no other components. Then (2) can be replaced by

$$(G_1 \setminus \Upsilon) \parallel G_2 \parallel \cdots \parallel G_n \ . \tag{4}$$

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 (Flordal and Malik, 2009). The abstraction is isomorphic to the original system apart from event renaming. An abstract counterexample for (4) may contain steps labelled τ that are not possible in the original system (2). 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 $\sim$, denoted $[x]$, 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 5. Let $G = \langle \Sigma, Q, \to, Q^\circ, Q^\omega \rangle$ 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 $G/\!\sim = \langle \Sigma, Q/\!\sim, \to/\!\sim, \tilde{Q}^\circ, \tilde{Q}^\omega \rangle$, where $\to/\!\sim \ = \{\, ([x], \sigma, [y]) \mid x \xrightarrow{\sigma} y \,\}$, $\tilde{Q}^\circ = \{\, [x^\circ] \mid x^\circ \in Q^\circ \,\}$, and $\tilde{Q}^\omega = \{\, [x^\omega] \mid x^\omega \in Q^\omega \,\}$.

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 (Flordal and Malik, 2009; Su et al., 2010). Therefore, it is common in compositional nonblocking verification to replace an FSM in (2) by its quotient and obtain an abstract system

$$(G_1/\!\sim) \parallel G_2 \parallel \cdots \parallel G_n \ . \tag{5}$$

If the abstracted system (5) 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 (2). The following condition ensures that this counterexample modification can be done using information about G_1 and $G_1/\!\sim$ only.

Definition 6. Let G and H be two FSMs. G is *counterexample-based less conflicting* than H, written $G \lesssim_{\text{ce}} H$, if for all paths $H \stackrel{s}{\Rightarrow} x_H$ there exists a state x_G of G such that $G \stackrel{s}{\Rightarrow} x_G$ and $\mathcal{L}^\omega(x_G) \subseteq \mathcal{L}^\omega(x_H)$.

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

Being counterexample-based less conflicting is a stronger property than being less conflicting, so this property can help to prove that an abstraction preserves conflict equivalence.

If Def. 6 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 $G \stackrel{s}{\Rightarrow} x_G$. If H is obtained by state merging, $H = G/\!\sim$, the condition $\mathcal{L}^\omega(x_G) \subseteq \mathcal{L}^\omega(x_H)$ can usually be replaced by $x_G \in x_H$, which is easier to implement.

Algorithm 1 is a search procedure to find a concrete counterexample when Def. 6 is satisfied after abstraction of G_1 in a composition (2). First, the loop in line 1 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. From line 3, the algorithm performs a search through the concrete FSM G_1 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 $(C, \tilde{C})$, where C is a partially constructed initial segment of a concrete counterexample, and $\tilde{C}$ is the remainder of the abstract counterexample still to be processed. The set *Visited* contains pairs (x_1, i) of concrete states x_1 of G_1 and positions i in the abstract counterexample, to ensure termination by avoiding duplicate search states. The loop in line 3 initialises the search with concrete counterexamples starting from each initial states of G_1. The main loop in line 7 decomposes each pair $(C, \tilde{C})$ as per lines 9–10 to get the matching state tuples at the end of C and the start of $\tilde{C}$, and explores transitions originating from the concrete end state x_1^i in G_1.

If the end of $\tilde{C}$ has been reached, line 12 checks the termination condition $\mathcal{L}^\omega(x_1^i) \subseteq \mathcal{L}^\omega(\tilde{x}_1^k)$ according to Def. 6, and returns C as the concrete counterexamples if satisfied. Otherwise, the next event σ_{i+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 14–20) or only performed by the rest of the system $G_2 \parallel \cdots \parallel G_n$ (lines 21–25). In any case, τ-transitions in G_1 also have to be explored (lines 26–30). The last loop should also include events present in G_1 but not in $G_2 \| \cdots \| G_n$, but such events will typically have been hidden and replaced by τ during compositional abstraction.

Assuming Def. 6, Algorithm 1 always terminates through line 13 before the *Queue* becomes empty. By processing pairs with shortest combined length first in line 8, the algorithm performs an A^**-search* that guarantees a shortest

Algorithm 1: State Merging Expansion

Input: $G_1, \ldots, G_n$ where $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_i^\circ, Q_i^\omega \rangle$
Input: Abstract counterexample $\tilde{C} = \tilde{y}^0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} \tilde{y}^k$ 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

1 **while** $\tilde{C}$ includes $\tilde{y}^i \xrightarrow{\tau} \tilde{y}^{i+1}$ where $\tilde{x}_1^i \neq \tilde{x}_1^{i+1}$ **do**
2 delete the first such transition together with its source state $\tilde{y}^i$ from $\tilde{C}$
3 **foreach** $x_1^\circ \in Q_1^\circ$ **do**
4 $C := (x_1^\circ, x_2^0, \ldots, x_n^0)$ *// 1-state counterexample*
5 add $(C, \tilde{C})$ to *Queue*
6 add $(x_1^\circ, 0)$ to *Visited*
7 **while** *Queue* is not empty **do**
8 remove $(C, \tilde{C})$ with minimal $|C| + |\tilde{C}|$ from *Queue*
9 let $\tilde{C} = \tilde{y}^i \xrightarrow{\sigma_{i+1}} \cdots \xrightarrow{\sigma_k} \tilde{y}^k$, $\tilde{y}^i = (\tilde{x}_1^i, x_2^i, \ldots, x_n^i)$
10 let $C = y^0 \rightarrow \cdots \rightarrow y^j$, $y^j = (x_1^i, x_2^i, \ldots, x_n^i)$
11 **if** $\tilde{C} = \tilde{y}^k$ (i.e., $\tilde{C}$ has only one state) **then**
12 **if** $\mathcal{L}^\omega(x_1^i) \subseteq \mathcal{L}^\omega(\tilde{x}_1^k)$ **then**
13 **return** C
14 **else if** $\sigma_{i+1} \in \Sigma_1$ **then**
15 **foreach** transition $x_1^i \xrightarrow{\sigma_{i+1}}_1 z_1$ in G_1 **do**
16 **if** $(z_1, i+1) \notin$ *Visited* **then**
17 $C' := C \xrightarrow{\sigma_{i+1}} (z_1, x_2^{i+1}, \ldots, x_n^{i+1})$
18 $\tilde{C}' := \tilde{y}^{i+1} \xrightarrow{\sigma_{i+2}} \cdots \xrightarrow{\sigma_k} \tilde{y}^k$
19 add $(C', \tilde{C}')$ to *Queue*
20 add $(z_1, i+1)$ to *Visited*
21 **else if** $(x_1^i, i+1) \notin$ *Visited* **then**
22 $C' := C \xrightarrow{\sigma_{i+1}} (x_1^i, x_2^{i+1}, \ldots, x_n^{i+1})$
23 $\tilde{C}' := \tilde{y}^{i+1} \xrightarrow{\sigma_{i+2}} \cdots \xrightarrow{\sigma_k} \tilde{y}^k$
24 add $(C', \tilde{C}')$ to *Queue*
25 add $(x_1^i, i+1)$ to *Visited*
26 **foreach** transition $x_1^i \xrightarrow{\tau}_1 z_1$ in G_1 **do**
27 **if** $(z_1, i) \notin$ *Visited* **then**
28 $C' := C \xrightarrow{\tau} (z_1, x_2^i, \ldots, x_n^i)$
29 add $(C', \tilde{C})$ to *Queue*
30 add (z_1, i) to *Visited*

result (Hart et al., 1968). Most state merging abstractions satisfy Def. 6 in such a way that the test $\mathcal{L}^\omega(x_1^i) \subseteq \mathcal{L}^\omega(\tilde{x}_1^k)$ in line 12 can be replaced by the simpler condition $x_1^i \in \tilde{x}_1^k$. This holds for *observation equivalence* (Milner, 1989) and *weak observation equivalence* (Su et al., 2010), and it can also be shown for other cases such as the *Active Events* and *Silent Continuation Rules* (Flordal and Malik, 2009).

The 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 in line 7 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 in lines 15 and 26. 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.

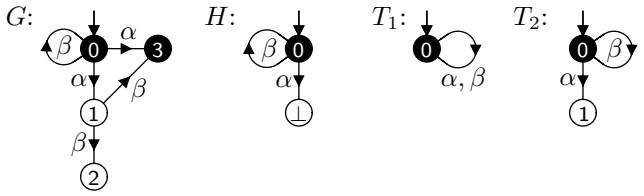

Fig. 1. Example of certain conflicts abstraction.

3.3 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* (Malik, 2004; Lindsey, 2012). FSMs G and H in Fig. 1 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 1, where β is needed to reach an accepting state, but β also leads to the blocking state 2. However, G is not counterexample-based less conflicting that H, 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. 6. Assume that the remainder of the system in Fig. 1 behaves like T_1, and the abstract counterexample for $H \parallel T_1$ is $(0,0) \xrightarrow{\alpha} (\bot, 0)$. Attempts to convert this to a concrete counterexample with end state $(1,0)$ or $(3,0)$ in $G \parallel T_1$ fail as these are not blocking states. A concrete counterexample can only be obtained by *extension*, e.g., $(0,0) \xrightarrow{\alpha} (1,0) \xrightarrow{\beta} (2,0)$ is a counterexample to the nonblocking property of $G \parallel T_1$. How to extend does not only depend on the abstracted FSM but also on the rest of the system (Lindsey, 2012). An abstract counterexample for $H \parallel T_2$ in Fig. 1 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* (Flordal and Malik, 2009) is considered, which is exemplary for all cases where Def. 6 does not apply.

Definition 7. (Limited Certain Conflicts Rule). Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an FSM. Define sets of *limited certain conflict states* inductively:

$$\mathrm{lcc}_G^0 = \{\, x \in Q \mid \mathcal{L}^\omega(x) = \emptyset \,\} ; \tag{6}$$

$$\mathrm{lcc}_G^{i+1} = \{\, x \in Q \mid \text{for every path } x = x_0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} x_k \in Q^\omega \text{ there exists } j \geq 0 \text{ such that } j \leq k \text{ and } x_j \stackrel{\varepsilon}{\Rightarrow} \mathrm{lcc}_G^i, \text{ or } j < k \text{ and } x_j \xRightarrow{\sigma_{j+1}} \mathrm{lcc}_G^i \,\} ; \tag{7}$$

$$\mathrm{lcc}_G = \bigcup_{i \geq 0} \mathrm{lcc}_G^i \, . \tag{8}$$

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

The set lcc_G^0 of *level-0* limited certain conflict states is the set of blocking states (6). Level $i+1$ adds to this states that can only reach accepting states by passing through

Algorithm 2: Limited Certain Conflicts Extension

Input: $G_1, \ldots, G_n$ where $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q_i^\circ, Q_i^\omega \rangle$
Input: Abstract counterexample $\tilde{C} = \tilde{y}^0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_k} \tilde{y}^k$ where $\tilde{y}^i = (x_1^i, x_2^i, \ldots, x_n^i)$ for $0 \leq i \leq k$
Output: Concrete counterexample

1 **if** $x_1^0 = \bot$ **then**
2 | $j := 0$; $I := (Q_1^\circ \cap \mathrm{lcc}_{G_1}) \times \{(x_2^0, \ldots, x_n^0)\}$
3 **else if** $x_1^i = \bot$ for some $1 \leq i \leq k$ **then**
4 | $j := \min\{\, i \mid x_1^i = \bot \,\} - 1$; $I := \{\tilde{y}^j\}$
5 **else**
6 | $j := k$; $I := \{\tilde{y}^k\}$
7 $m := \min\{\, i \mid \mathrm{lcc}_{G_1}^i = \mathrm{lcc}_{G_1} \,\}$; $Q' := Q_2 \times \cdots \times Q_n$
8 **while** $m \geq 0$ **and** $I \rightarrow \mathrm{lcc}_{G_1}^m \times Q'$ in $G_1 \parallel \cdots \parallel G_n$ **do**
9 | assign E to be the path $I \rightarrow \mathrm{lcc}_{G_1}^m \times Q'$
10 | $m := m - 1$
11 **if** E is unassigned **then**
12 | **return** $\tilde{C}$
13 **else if** $j = 0$ **then**
14 | **return** E
15 **else**
16 | **return** $\tilde{y}^0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_j} E$

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 (7). 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. In Fig. 1, for example, $\mathrm{lcc}_G^0 = \{2\}$ 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).

Proposition 2. Let G be an FSM. Then $G \simeq_{\mathrm{conf}} \textsc{Lcc}(G)$.

By Prop. 2, during compositional nonblocking verification, an FSM G_1 in (2) can be abstracted to get

$$\textsc{Lcc}(G_1) \parallel G_2 \parallel \cdots \parallel G_n \ . \tag{9}$$

An abstract counterexample for (9) is accepted by all the FSMs in the original system (2), 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 $\mathrm{lcc}_{G_1}^i$ (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 (9), 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 (line 2), otherwise from the last state before $\bot$ in $\tilde{C}$ (line 4). 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 $\tilde{C}$ (line 6).

Once the set I of start states is determined, the loop in line 8 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 (2). If no extension can be found, the abstract counterexample is returned unchanged (line 12). Otherwise the result is the extension, possibly preceded by the steps of the abstract counterexample that do not include $\bot$ (line 14 or 16).

From Def. 7 it follows that (i) an abstract counterexample not into $\bot$ is a concrete counterexample, or it can be extended into limited certain conflicts; and (ii) a concrete path into limited certain conflicts is a concrete counterexample, or it can be extended to a lower level of limited certain conflicts. These observations are the basis for the correctness proof of Algorithm 2.

Proposition 3. Let $G_1, \ldots, G_n$ be FSMs. If $\tilde{C}$ is a counterexample to the nonblocking property of $\textsc{Lcc}(G_1) \| G_2 \| \cdots \| G_n$, then Algorithm 2 terminates and returns a counterexample to the nonblocking property of $G_1 \parallel \cdots \parallel G_n$.

The search for the extension in line 8 can be done with a *language inclusion check* (Ware and Malik, 2008). It involves the full composed state space of (2), which may not be feasible to explore in the context of compositional verification. One option is to use the *iterative projection algorithm* (Ware and Malik, 2008), which has similar performance characteristics to compositional nonblocking verification.

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* (Wirth, 1986), 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.

4. EXPERIMENTAL RESULTS

The counterexample expansion procedure is part of the compositional conflict check in Supremica (Åkesson et al., 2006). It has been used to compute counterexamples for 21 examples. The test suite includes complex industrial models and case studies from different application areas such as manufacturing systems, automotive electronics, and communication protocols (Pilbrow and Malik, 2015).

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, weak observation equivalence, and marking saturation (Flordal and Malik, 2009; Pilbrow and Malik, 2015).

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 #).

Table 1. Experimental results.

Model				No LCC			LCC				
		State	CE	Total	Exp	CE	Total	Exp	Ext	Ext	CE
Name	n	space	min	[s]	[s]	len	[s]	[s]	[s]	#	len
agvb	17	$2.3 \cdot 10^7$	6	0.2	0.0	18	0.3	0.0	0.0	4	17
aip0alps	35	$3.0 \cdot 10^8$	4	0.3	0.0	19	0.3	0.0	0.1	1	20
aip0tough	60	$1.0 \cdot 10^{10}$	39	7.1	0.1	149	7.1	0.1	6.4	5	39
aip1efa ⟨16⟩	50	$9.5 \cdot 10^{12}$	153	35.5	0.2	185	36.0	0.6	1.3	1	185
aip1efa ⟨24⟩	50	$1.8 \cdot 10^{13}$	153	27.9	0.1	185	27.7	0.1	0.0	0	185
ct17	67	$3.9 \cdot 10^{22}$	3	0.5	0.0	5	0.3	0.0	0.0	1	5
fencaiwon09b	31	$8.9 \cdot 10^7$	225	0.4	0.0	249	0.4	0.0	0.0	1	249
fms2003	30	$1.7 \cdot 10^7$	20	1.4	0.1	34	0.4	0.0	0.0	1	20
ftechnik	36	$1.2 \cdot 10^8$	0	0.4	0.0	3	0.5	0.0	0.1	3	3
prime_sieve4b	16	$1.2 \cdot 10^{20}$	31	5.6	0.4	32	7.5	0.5	3.5	10	32
psl_partleft	39	$7.7 \cdot 10^7$	4	6.0	0.1	9	0.6	0.0	0.4	3	15
psl_restart	37	$3.9 \cdot 10^7$	8	1.1	0.1	50	0.9	0.0	0.3	4	25
tbed_ctct	84	$3.9 \cdot 10^{13}$	0	7.0	0.1	0	434.3	0.3	428.5	11	203
tbed_hisc1	184	$2.9 \cdot 10^{17}$	19	2.8	0.1	22	2.1	0.1	0.9	10	31
tbed_noderailb	84	$3.2 \cdot 10^{12}$	2	3.4	0.2	4	43.6	0.2	40.4	8	4
tip3_bad	54	$5.2 \cdot 10^{10}$	16	0.8	0.1	24	0.8	0.1	0.0	0	24
verriegel3b	52	$1.3 \cdot 10^9$	4	0.7	0.0	53	0.8	0.0	0.4	3	4
verriegel4b	64	$6.3 \cdot 10^{10}$	4	1.0	0.1	81	1.0	0.0	0.5	3	4
6linka	53	$2.4 \cdot 10^{14}$	5	0.3	0.0	6	0.3	0.0	0.0	1	16
6linkp	48	$4.4 \cdot 10^{14}$	1	0.4	0.0	11	0.2	0.0	0.0	1	11
6linkre	59	$6.2 \cdot 10^{13}$	90	0.4	0.0	102	0.5	0.1	0.1	11	107

The compositional conflict check algorithm proves all these models to be blocking within seconds. While the expansion time of Algorithm 1 is insignificant, 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** and **tbed_hisc1** 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. CONCLUSIONS

Two algorithms for counterexample computation for different abstraction rules during compositional nonblocking verification are proposed. While counterexample *expansion* is fast and simple, some abstraction rules require a more time-consuming algorithm of counterexample *extension*. The counterexample computation process depends on the type of abstraction used during verification, and this paper covers the most common rules (Flordal and Malik, 2009). More advanced abstraction (Ware and Malik, 2012; Pilbrow and Malik, 2015) will be studied in future work.

REFERENCES

Åkesson, K., Fabian, M., Flordal, H., and Malik, R. (2006). Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In *8th Int. Workshop on Discrete Event Systems, WODES '06*, 384–385. IEEE. doi:10.1109/WODES.2006.382401.

Baier, C. and Katoen, J.P. (2008). *Principles of Model Checking*. MIT Press.

De Nicola, R. and Hennessy, M.C.B. (1984). Testing equivalences for processes. *Theoretical Comput. Sci.*, 34(1–2), 83–133. doi:10.1016/0304-3975(84)90113-0.

Flordal, H. and Malik, R. (2009). Compositional verification in supervisory control. *SIAM J. Control Optim.*, 48(3), 1914–1938. doi:10.1137/070695526.

Graf, S. and Steffen, B. (1990). Compositional minimization of finite state systems. In *1990 Workshop on Computer-Aided Verification*, volume 531 of *LNCS*, 186–196. Springer. doi:10.1007/BFb0023732.

Hart, P.E., Nilsson, N.J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2), 100–107. doi:10.1109/TSSC.1968.300136.

Hoare, C.A.R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

Lindsey, J. (2012). The set of certain conflicts. Honours project report, Dept. of Computer Science, University of Waikato.

Malik, R. (2004). On the set of certain conflicts of a given language. In *7th Int. Workshop on Discrete Event Systems, WODES '04*, 277–282. IFAC. doi:10.1016/S1474-6670(17)30757-7.

Malik, R. (2016). Programming a fast explicit conflict checker. In *13th Int. Workshop on Discrete Event Systems, WODES '16*, 464–469. IEEE. doi:10.1109/WODES.2016.7497885.

Malik, R., Streader, D., and Reeves, S. (2006). Conflicts and fair testing. *Int. J. Found. Comput. Sci.*, 17(4), 797–813. doi:10.1142/S012905410600411X.

Milner, R. (1989). *Communication and concurrency*. Series in Computer Science. Prentice-Hall.

Pena, P.N., Cury, J.E.R., and Lafortune, S. (2009). Verification of nonconflict of supervisors using abstractions. *IEEE Trans. Autom. Control*, 54(12), 2803–2815. doi:10.1109/TAC.2009.2031730.

Pilbrow, C. and Malik, R. (2015). An algorithm for compositional nonblocking verification using special events. *Sci. Comput. Programming*, 113(2), 119–148. doi:10.1016/j.scico.2015.05.010.

Ramadge, P.J.G. and Wonham, W.M. (1989). The control of discrete event systems. *Proc. IEEE*, 77(1), 81–98. doi:10.1109/5.21072.

Su, R., van Schuppen, J.H., Rooda, J.E., and Hofkamp, A.T. (2010). Nonconflict check by using sequential automaton abstractions based on weak observation equivalence. *Automatica*, 46(6), 968–978. doi:10.1016/j.automatica.2010.02.025.

Ware, S. and Malik, R. (2008). The use of language projection for compositional verification of discrete event systems. In *9th Int. Workshop on Discrete Event Systems, WODES '08*, 322–327. IEEE. doi:10.1109/WODES.2008.4605966.

Ware, S. and Malik, R. (2012). Conflict-preserving abstraction of discrete event systems using annotated automata. *Discrete Event Dyn. Syst.*, 22(4), 451–477. doi:10.1007/s10626-012-0133-3.

Wirth, N. (1986). *Algorithms and Data Structures*. Prentice-Hall.