Counterexample Computation in Compositional Nonblocking Verification

Robi Malik ∗ Simon Ware ∗∗

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 τ ∉ Σ labels transitions that are only taken by the component under consideration. Σ* denotes the set of all finite traces of the form σ₁σ₂ · · · σₙ of events from Σ, including the empty trace ε. The concatenation of two traces s, t ∈ Σ* is written as st. A subset L ⊆ Σ* is called a language. The standard projection P: (Σ ∪ {τ})* → Σ* is the operation that deletes all silent (τ) events from traces.

Definition 1. A (nondeterministic) finite-state machine (FSM) is a tuple G = ⟨Σ, Q, →, Q0, Qc⟩ where Σ is a set of events, Q is a finite set of states, → ⊆ Q × (Σ ∪ {τ}) × Q is the transition relation, Q0 ⊆ Q is the set of initial states, and Qc ⊆ Q is the set of accepting states.

The transition relation is written in infix notation x → y and extended to traces s ∈ (Σ ∪ {τ})* in the standard way. For state sets X, Y ⊆ Q, the notation X → Y means x → y for some x ∈ X and y ∈ Y. For states or state sets x and y, the notation x → y means x → y for some s ∈ (Σ ∪ {τ})*, and x → z means x → z for some z ∈ Q.
Events not in the event set of an FSM are always enabled without state change, so the transition relation is further extended by \( x \xrightarrow{\tau} y \) for all \( x \in Q \) and \( \tau \notin \Sigma \cup \{\tau\} \).

To support silent events, another transition relation \( \Rightarrow \subseteq Q \times \Sigma^* \times Q \) is introduced, where \( x \xrightarrow{\sigma} y \) denotes the existence of a trace \( t \in (\Sigma \cup \{\tau\})^* \) such that \( P(t) = s \) and \( x \xrightarrow{\sigma} y \). That is, \( x \xrightarrow{\sigma} y \) denotes a path with exactly the events in \( s \), while \( x \xrightarrow{\tau} y \) denotes a path with an arbitrary number of \( \tau \) events shuffled with the events of \( s \).

Notations such as \( X \xrightarrow{\sigma} Y \) and \( X \xrightarrow{\tau} Y \) are defined analogously to \( \Rightarrow \). The accepting language of a state or state set \( x \) is \( \mathcal{L}(x) = \{ s \in \Sigma^* | x \xrightarrow{\sigma} Q^o \} \). For an FSM \( G \), the notation \( G \xrightarrow{\sigma} x \) means \( Q^o \xrightarrow{\sigma} x \).

FSMs are synchronised in lock-step (Hoare, 1985). The synchronous composition of two FSMs \( G = (\Sigma_G, Q_G, \rightarrow_G, Q^G_0, Q^G_f) \) and \( H = (\Sigma_H, Q_H, \rightarrow_H, Q^H_0, Q^H_f) \) is

\[
G \parallel H = (\Sigma_G \cup \Sigma_H, Q_G \times Q_H, \rightarrow, Q^G_0 \times Q^H_0, Q^G_f \times Q^H_f)
\]

where

\[
(x_G, x_H) \xrightarrow{\sigma} (y_G, y_H) \text{ if } \sigma \neq \tau, \quad x_G \xrightarrow{\sigma} G y_G, \quad x_H \xrightarrow{\tau} H y_H;
\]

\[
(x_G, x_H) \xrightarrow{\tau} (y_G, x_H) \text{ if } x_G \xrightarrow{\tau} G y_G;
\]

\[
(x_G, x_H) \xrightarrow{x_G, y_H} (y_G, y_H) \text{ if } x_H \xrightarrow{\tau} H y_H.
\]

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 \( G = (\Sigma, Q, \rightarrow, Q^o, Q^f) \) is nonblocking if, for every state \( x \in Q \) and every trace \( s \in \Sigma^* \) such that \( Q^o \xrightarrow{s} x \), there exists a trace \( t \in \Sigma^* \) such that \( x \xrightarrow{t} Q^f \); 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 \( G = (\Sigma, Q, \rightarrow, Q^o, Q^f) \) be an FSM. A counterexample to the nonblocking property of \( G \) is a path

\[
x_0 \xrightarrow{\sigma_1} x_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} x_n
\]

such that \( x_0 \in Q^o \), and \( \mathcal{L}'(x_n) = \emptyset \).

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, \( \mathcal{L}'(x_n) = \emptyset \), or equivalently \( x_n \xrightarrow{\tau} Q^f \) 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 conflict than \( H \), written \( G \leq_{\text{conf}} H \), if for any FSM \( T \) such that \( G \parallel T \) is nonblocking, it also holds that \( H \parallel T \) is nonblocking. \( G \) is conflict equivalent, \( G \equiv_{\text{conf}} H \), if \( G \leq_{\text{conf}} H \) and \( H \leq_{\text{conf}} G \).

When verifying whether a composed system of FSMs \( G_1 \parallel G_2 \parallel \cdots \parallel G_n \), \( n \geq 2 \), 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 sub-system \( (G_j \in J) \) 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 \( (G_i \in J) \). The congruence properties (Malik et al., 2006) of conflict equivalence ensure that \( G \) is nonblocking if and only if the original system \( (G_i \in J) \) is. \( G \) typically has fewer states than \( (G_i \in J) \), making it possible to check whether it is nonblocking even though the full composition \( (G_i \in J) \) may be too large to fit in memory.

3. COUNTEREXAMPLE EXPANSION

Assume that a system \( (G_i \in J) \) 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 \( (G_i \in J) \).

In the following, a counterexample to the nonblocking property of an abstracted system is called an abstract counterexample, while a counterexample to the blocking 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 \( (G_i \in J) \) 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 \( (G_i \in J) \) can be replaced by

\[
(G_1 \parallel G_2 \parallel \cdots \parallel G_n)
\]
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 $\Upsilon$ appear only in $G_1$ and in no other components. Then (2) can be replaced by

$$(G_1 \setminus \Upsilon) \parallel G_2 \parallel \cdots \parallel G_n.$$  \hspace{1cm} (4)

Here, $G_1 \setminus \Upsilon$ is the result of hiding, which is obtained from $G_1$ by replacing all events in $\Upsilon$ by the silent event $\tau$ (Flordal and Malik, 2009). The abstraction is isomorphic to the original system apart from event renaming. An abstract counterexample for (4) may contain states labelled $\tau$ that are not possible in the original system (2). These steps can be identified from the state information, and their $\tau$ 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 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 $G = (\Sigma, Q, \rightarrow, Q^0, Q^\omega)$ be an FSM, and let $\sim \subseteq Q \times Q$ be an equivalence relation. The quotient FSM $G/\sim$ of $G$ with respect to $\sim = (\Sigma, Q/\sim, \rightarrow/\sim, Q^0/\sim, Q^\omega/\sim)$, where $\rightarrow/\sim = \{ ([x], [y]) \mid x \rightarrow y \}$, $Q^0/\sim = \{ [x^0] \mid x^0 \in Q^0 \}$, and $Q^\omega/\sim = \{ [x^\omega] \mid x^\omega \in Q^\omega \}$.

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 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

$$(G_1/\sim) \parallel G_2 \parallel \cdots \parallel G_n.$$  \hspace{1cm} (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 $G \preceq C H$, if for all paths $H \Rightarrow x_H$ there exists a state $x_G$ of $G$ such that $G \Rightarrow x_G$ and $L^c(x_G) \subseteq L^c(x_H)$.

**Proposition 1.** Let $G$ and $H$ be two FSMs. If $G \preceq C H$ then $G \preceq \text{conf } H$.

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., $H \Rightarrow x_H$, 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 $x_H$ while using the same non-$\tau$ events, which must be possible because $G \Rightarrow x_G$. If $H$ is obtained by state merging, $H = G/\sim$, the condition $L^c(x_G) \subseteq L^c(x_H)$ 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 $\tau$-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 $\tau$-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 $\text{Visited}$ contains pairs $(x_i, i)$ of concrete states $x_i$ 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 state 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 $x'_1$ in $G_1$.

If the end of $\tilde{C}$ has been reached, line 12 checks the termination condition $L^c(x'_1) \subseteq L^c(x_1)$ according to Def. 6, and returns $C$ as the concrete counterexamples if satisfied. Otherwise, the next event $\sigma_{x'_1}$ 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, $\tau$-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 $G_2 \parallel \cdots \parallel G_n$, but such events will typically have been hidden and replaced by $\tau$ 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 $G_i = \langle \Sigma_i, Q_i, \rightarrow_i, Q^p_i, Q^c_i \rangle$

**Input:** Abstract counterexample $\hat{C} = \hat{y}^0 \rightarrow \hat{y}^1 \rightarrow \cdots \hat{y}^k$ where $\hat{y}^i = (\hat{x}_1^i, x_2^i, \ldots, x_n^i)$ for $0 \leq i \leq k$

**Output:** Concrete counterexample $C$

1. **while** $\hat{C}$ includes $\hat{y}^i \rightarrow \hat{y}^{i+1}$ where $\hat{x}_1^i \neq \hat{x}_1^{i+1}$ **do**
   1. delete the first such transition together with its source state $\hat{y}^i$ from $\hat{C}$
   2. **foreach** $x_1^i \in Q^i_1$ **do**
      1. $C := (x_1^i, x_2^i, \ldots, x_n^i) \ // 1$-state counterexample
      2. add $(C, C)$ to Queue
      3. add $(x_1^i, 0)$ to Visited
   3. **while** Queue is not empty do
      1. remove $(C, C)$ with minimal $|C| + |\hat{C}|$ from Queue
      2. let $\hat{C} = C = \hat{y}^0 \rightarrow \cdots \hat{y}^k$, $\hat{y}^i = (\hat{x}_1^i, x_2^i, \ldots, x_n^i)$
      3. let $C = \hat{y}^k \rightarrow \cdots \rightarrow y^i$, $y^i = (x_1^i, x_2^i, \ldots, x_n^i)$
   4. if $C = \hat{y}^k$ (i.e., $\hat{C}$ has only one state) **then**
      1. **if** $L^C(x_1^i) \subseteq L^\hat{C}(\hat{x}_1^i)$ **then**
          1. return $C$
      2. else if $\sigma_{i+1} \in \Sigma_i$ **then**
          1. **foreach** transition $x_1^i \xrightarrow{\sigma_{i+1}} z_1$ in $G_1$ **do**
              1. **if** $(z_1, i+1) \notin Visited$ **then**
                  1. $C' := C \xrightarrow{\sigma_{i+1}} (z_1, x_2^{i+1}, \ldots, x_n^{i+1})$
                  2. add $(C', C)$ to Queue
                  3. add $(z_1, i+1)$ to Visited
          2. else if $(x_1^i, i+1) \notin Visited$ **then**
              1. $C' := C \xrightarrow{\sigma_{i+2}} (x_1^i, x_2^{i+1}, \ldots, x_n^{i+1})$
              2. $\hat{C}' := \hat{y}^{i+1} \rightarrow \hat{y}^k$
              3. add $(C', C')$ to Queue
              4. add $(x_1^i, i+1)$ to Visited
          3. **foreach** transition $x_1^i \xrightarrow{\hat{C}} z_1$ in $G_1$ **do**
              1. **if** $(z_1, i) \notin Visited$ **then**
                  1. $C' := C \xrightarrow{\hat{C}} (z_1, x_2^i, \ldots, x_n^i)$
                  2. add $(C', C)$ to Queue
                  3. 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 $L^C(x_1^i) \subseteq L^\hat{C}(\hat{x}_1^i)$ in line 12 can be replaced by the simpler condition $x_1^i \subset \hat{x}_1^i$. 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 $|\hat{C}| + |Q_1|$, 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 $O(|\hat{C}||Q_1|^3)$, 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 $\alpha$ is conflicting with both $G$ and $H$. Note that execution of $\alpha$ may take $G$ to state 1, where $\beta$ is needed to reach an accepting state, but $\beta$ also has the blocking state 2. However, $G$ is not counterexample-based less conflicting that $H$, because for $H \models \bot$ with $L^\bot(\bot) = \emptyset$, there is no state in $G$ reachable via $\alpha$ 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 $(0,0) \not\models (\bot,0)$. 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., $(0,0) \not\models (1,0) \not\models (2,0)$ 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 $(0,0) \not\models (\bot,1)$, 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 $G = \langle \Sigma, Q, \rightarrow, Q^p, Q^c \rangle$ be an FSM. Define sets of limited certain conflict states inductively:

$$\text{lcc}_G^0 = \{ x \in Q \mid L^C(x) = \emptyset \}$$

(6)

$$\text{lcc}_G^{i+1} = \{ x \in Q \mid \text{for every path } x = x_0 \rightarrow \cdots \rightarrow x_k \text{ there exists } j \geq 0 \text{ such that } j \leq k \text{ and } x_j \not\models \text{lcc}_G \}$$

(7)

$$\text{lcc}_G = \bigcup_{i \geq 0} \text{lcc}_G^i$$

(8)

The limited certain conflicts abstraction of $G$ is $Lcc(G) = \langle \Sigma, Q_{lcc}, \rightarrow_{lcc}, Q^p_{lcc}, Q^c_{lcc} \rangle$ where $Q_{lcc} = (Q \setminus \text{lcc}_G) \cup \{ \bot \}$ (with $\bot \notin Q$); $x \not\models_{lcc} y$ if $x, y \neq \bot$ and $x \not\models y$ and $x \not\models_{lcc} \text{lcc}_G$ does not hold, or $x \neq \bot = y$ and $x \not\models_{lcc} \text{lcc}_G$; $Q^p_{lcc} = Q^p$ if $Q^p \cap \text{lcc}_G = \emptyset$ and $Q^c_{lcc} = \{ \bot \}$ otherwise; and $Q^c_{lcc} = Q^c \setminus \text{lcc}_G$.

The set $\text{lcc}_G^i$ of level-$i$ 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 $G_i = (\Sigma_i, Q_i, \rightarrow_i, Q^0_i, \Sigma^i)$

Input: Abstract counterexample $\tilde{C} = \tilde{y}_0 \tilde{a}_1 \ldots \tilde{y}_k$

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 $x_0^1 = \perp$ then
2. $j := 0; I := (Q^1_1 \cap \text{lcc } G_1) \times \{(x_2^0, \ldots, x_n^0)\}$
3. else if $x_1^1 = \perp$ for some $1 \leq i \leq k$ then
4. $j := \min \{ i \mid x_1^i = \perp \} - 1; I := \{(\tilde{y}_i)\}$
5. else
6. $i \leftarrow k; I := (\tilde{y}_k)$
7. $m := \min \{ i \mid \text{lcc } G_i = \text{lcc } G_1 \}; Q' := Q_2 \times \cdots \times Q_n$
8. while $m \geq 0$ and $I \rightarrow \text{lcc } G_m \times Q'$ in $G_1 \parallel \cdots \parallel G_n$
9. do $I \leftarrow \text{lcc } G_m \times Q'$
10. assign $E$ to be the path $\tilde{I} \rightarrow \text{lcc } G_m \times Q'$
11. $m := m - 1$
12. if $E$ is unassigned then
13. return $\tilde{C}$
14. else if $j = 0$ then
15. return $E$
16. else return $\tilde{y}_0 \tilde{a}_1 \ldots \tilde{a}_k E$

a state that can reach a level-$i$ limited certain conflict state using $\tau$-transitions, or using a transition that may lead to a level-$i$ state (7). These sets form an increasing sequence, $\text{lcc } G_m \subseteq \text{lcc } G_{m-1} \subseteq \cdots$, which in the finite-state case converges against the set $\text{lcc } G$. The abstraction $\text{lcc } G(G)$ is constructed by merging these sets into a new state $\perp$, and deleting some transitions. In Fig. 1, for example, $\text{lcc } G_2 = \{2\}$ and $\text{lcc } G_3 = \text{lcc } G_4 = \{1, 2\}$ for $i \geq 1$. This results in the abstraction $\text{lcc } G(G) = H$ (the unreachable state 3 is not shown in the figure).

Proposition 2. Let $G$ be an FSM. Then $G \approx_{\text{conf}} \text{lcc } G(G)$.

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

$$\text{lcc } G(G_1) \parallel G_2 \parallel \cdots \parallel G_n.$$ (9)

An abstract counterexample for (9) is accepted by all the FSMS in the original system (2), up to the point where $\text{lcc } G(G_1)$ visits the new state $\perp$. If $\perp$ 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 $\text{lcc } G_2$, (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 $\perp$, then the search starts from the initial states of the concrete FSM $G_1$ (line 2), otherwise from the last state before $\perp$ in $\tilde{C}$ (line 4).

If the abstract counterexample does not contain $\perp$ at all, extension may still be needed as $G_1$ could reach an accepting state with transitions removed in $\text{lcc } G(G_1)$. 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 $\perp$ (line 14 or 16).

From Def. 7 it follows that (i) an abstract counterexample not into $\perp$ 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 $\text{lcc } G(G_1) \parallel G_2 \parallel \cdots \parallel G_n$, then Algorithm 2 terminates and returns a counterexample to the nonblocking property of $G_1 \parallel \cdots \parallel G_n$.

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 be feasible to explore in the presence 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 (Akesson et al., 2006). It has been used to compute counterexamples for 21 examples. The test suite includes simple 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 $\tau$-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 #).
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 \texttt{tbed}\_ctct and \texttt{tbed}\_noderailib tests, by far outweighing the small verification time benefit from the improved abstraction. In other cases such as \texttt{psl}\_partleft and \texttt{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., \texttt{aip0tough} and \texttt{verriegl41b}, limited certain conflicts lead to shorter counterexamples, while in other cases such as \texttt{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 \textit{expansion} is fast and simple, some abstraction rules require a more time-consuming algorithm of counterexample \textit{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


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


