Modular Nonblocking Verification Using Conflict Equivalence

Hugo Flordal
Department of Signals and Systems
Chalmers University of Technology, Gothenburg, Sweden
flordal@chalmers.se

Robi Malik
Department of Computer Science
University of Waikato, Hamilton, New Zealand
robi@cs.waikato.ac.nz

Abstract— **This paper proposes a modular approach to verifying whether a large discrete event system is nonconflicting. The new approach avoids computing the synchronous product of a large set of finite-state machines. Instead, the synchronous product is computed gradually, and intermediate results are simplified using conflict-preserving abstractions based on process-algebraic results about fair testing. Heuristics are used to choose between different possible abstractions. Experimental results show that the method is applicable to finite-state machine models of industrial scale and brings considerable improvements in performance over other methods.**

I. Introduction

Supervisory control theory for discrete event systems [1], [2] is a general framework for the design of supervisors for reactive systems. Given a model of the physically possible behaviour of the system to be controlled, the *plant*, and a *specification* of the desired system behaviour, the framework provides methods and algorithms to obtain a *supervisor* that ensures that the specification is always fulfilled. Two properties are commonly expected from such a supervisor: it should be *controllable* and *nonblocking*. Controllability typically captures safety requirements, while nonblocking is a special kind of liveness property.

Since supervisory control is used for complex, safety-critical systems, it is crucial to verify automatically that the supervisor satisfies the required properties. Yet, all basic algorithms for verification suffer greatly from the so-called *state-space explosion problem*. The problem is caused by explicitly describing the system's state space in a *monolithic* fashion. Typically, the model of the system, including the supervisor, is *modular*, and this feature can be exploited to perform verification more efficiently. For controllability, efficient solutions are presented in [3], [4], while nonblocking has long remained a challenge.

To enable modular reasoning about nonblocking, it is convenient to have a means of expressing equivalence between automata in this respect. Such an equivalence, *conflict equivalence*, was presented in [5]. Essentially, two automata are conflict equivalent if they show the same behaviour with respect to conflicts together with any other automaton.

This paper proposes a set of reduction procedures that can simplify finite-state automata in a way that preserves conflict equivalence. Using these reductions, the authors have implemented an algorithm for determining whether large models of discrete events systems are nonblocking in the DES software tool Supremica [**?**]. After introducing the required notation in section II, this paper explains the reduction procedures for conflict equivalence in section III. Afterwards, section IV describes the algorithm, and section V discusses the experimental results on a set of complex examples. Finally, section VI gives some concluding remarks.

II. Preliminaries

A. Events and Languages

Event sequences and languages are a simple means to describe discrete event system behaviours. Their basic building blocks are *events*, which are taken from a finite *alphabet*. In addition, two special events are also used, the *silent event* τ and the *termination event* ω. These are never included in an alphabet Σ unless mentioned explicitly. For this, $\Sigma_\tau = \Sigma \cup \{\tau\}$ and $\Sigma_{\tau,\omega} = \Sigma \cup \{\tau, \omega\}$ are used.

Σ^* denotes the set of all finite *strings* of the form $\sigma_1\sigma_2 \cdots \sigma_n$ of events from Σ, including the *empty string* ε. A subset $\mathcal{L} \subseteq \Sigma^*$ is called a *language*. The *catenation* of two strings $s, t \in \Sigma^*$ is written as st. Languages and alphabets can also be catenated, $\mathcal{L}\Sigma = \{\, s\sigma \mid s \in \mathcal{L}, \sigma \in \Sigma \,\}$.

B. Nondeterministic Automata

System behaviours are modelled using finite-state automata. Typically, system models are deterministic, but abstraction during the verification process may result in nondeterminism.

Definition 1: A (nondeterministic) *finite-state automaton* is a 5-tuple $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ where Q is a finite set of *states*, Σ is a finite set of *events*, $\rightarrow \subseteq Q \times \Sigma_\tau \times Q$ is the *state transition relation*, $Q^i \subseteq Q$ is the (nonempty) set of *initial states*, and $Q^m \subseteq Q$ is the set of *marked states*. ■

The transition relation is written in infix notation $p \xrightarrow{\sigma} q$, and is extended to strings in Σ_τ^* by letting

$$p \xrightarrow{\varepsilon} p \qquad \text{for all } p \in Q \ ; \tag{1}$$
$$p \xrightarrow{s\sigma} q \qquad \text{if } p \xrightarrow{s} r \text{ and } r \xrightarrow{\sigma} q \text{ for some } r \in Q \ . \tag{2}$$

For state sets $Q_1, Q_2 \subseteq Q$, the notation $Q_1 \xrightarrow{s} Q_2$ denotes the existence of $q_1 \in Q_1$ and $q_2 \in Q_2$ such that $q_1 \xrightarrow{s} q_2$. Furthermore, $p \rightarrow q$ denotes the existence of a string $s \in \Sigma_\tau^*$ such that $p \xrightarrow{s} q$, and $p \xrightarrow{s}$ denotes the existence of a state q such that $p \xrightarrow{s} q$. A state q is called *reachable* in an automaton G if $Q^i \rightarrow q$; if this holds for all $q \in Q$, G is called *accessible*. If G is not accessible, it can easily be made so by removing all states that are not reachable. Therefore, in the following all automata are assumed to be accessible.

For brevity, $p \overset{s}{\Rightarrow} q$, with $s = \sigma_1\sigma_2 \ldots \sigma_n$, is introduced to denote the existence of a string $t \in \tau^*\sigma_1\tau^*\sigma_2\tau^* \cdots \tau^*\sigma_n\tau^*$ such that $p \xrightarrow{t} q$. That is, $\xrightarrow{s}$ denotes a path with *exactly* the events in s while $\overset{s}{\Rightarrow}$ denotes a path with an arbitrary number of τ shuffled with the events in s. Moreover, G is said to *accept* a string $s \in \Sigma^*$ if $Q^i \overset{s}{\Rightarrow}$, and for a state $q \in Q$, the set of *active events* is defined as $\Sigma(q) = \{\, \sigma \in \Sigma \mid q \overset{\sigma}{\Rightarrow} \}$.

Automata use the set Q^m of marked states to indicate the possibility of successful termination. In order to translate this into an event-based representation, every automaton is assumed to have a terminal state $\perp \in Q \setminus Q^m$, which has no outgoing transitions, and the transition relation is extended to a relation $\rightarrow \subseteq Q \times \Sigma_{\tau,\omega} \times Q$ by adding transitions

$$q^m \xrightarrow{\omega} \perp \quad \text{for each} \quad q^m \in Q^m \ . \tag{3}$$

This construction makes it possible to represent termination by means of the termination event ω only which, if it occurs, is always the last event of any execution.

Definition 2: An automaton $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ is *deterministic* if Q^i is a singleton, $p \xrightarrow{\sigma} q_1$ and $p \xrightarrow{\sigma} q_2$ always implies $q_1 = q_2$, and $\rightarrow$ contains no τ-transitions. ■

When two automata are running in parallel, lock-step synchronisation in the style of [6] is used.

Definition 3: Let $G_1 = \langle Q_1, \Sigma_1, \rightarrow_1, Q_1^i, Q_1^m \rangle$ and $G_2 = \langle Q_2, \Sigma_2, \rightarrow_2, Q_2^i, Q_2^m \rangle$ be two automata. The *synchronous composition* of G_1 and G_2 is

$$G_1 \| G_2 \ = \ \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \rightarrow, Q_1^i \times Q_2^i, Q_1^m \times Q_2^m \rangle \tag{4}$$

where

$$\begin{aligned}
&(p,q) \xrightarrow{\sigma} (p',q') \text{ if } \sigma \in (\Sigma_1 \cap \Sigma_2) \cup \{\omega\},\ p \xrightarrow{\sigma}_1 p',\ q \xrightarrow{\sigma}_2 q' \,;\\
&(p,q) \xrightarrow{\sigma} (p',q) \text{ if } \sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau\},\ p \xrightarrow{\sigma}_1 p' \,;\\
&(p,q) \xrightarrow{\sigma} (p,q') \text{ if } \sigma \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau\},\ q \xrightarrow{\sigma}_2 q' \,.
\end{aligned}$$
■

In synchronous composition, shared events (and ω) must be executed by all automata synchronously, while other events (including τ) are executed independently.

Hiding is the act of transforming an event σ into a silent τ event. This is a simple way of abstraction that in general introduces nondeterminism.

Definition 4: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ and $\Upsilon \subseteq \Sigma$. The result of *hiding* Υ from G, written $G \backslash \Upsilon$, is the automaton obtained from G by replacing each transition $p \xrightarrow{\upsilon} q$ with $\upsilon \in \Upsilon$ by $p \xrightarrow{\tau} q$, and removing the events in Υ from Σ. ■

C. Conflicts

In supervisory control theory, an important property for an automaton is to be able to, from all reachable states, reach some marked state. If an automaton satisfies this property it is called *nonblocking*, otherwise *blocking*. When more than one automaton is involved, it also is common to use the terms *nonconflicting* and *conflicting*, respectively.

Definition 5: An automaton $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ is *nonblocking* if, for every state $q \in Q$ and every string $s \in \Sigma^*$ such that $Q^i \overset{s}{\Rightarrow} q$, there exists a string $t \in \Sigma^*$ such that $q \overset{t\omega}{\Rightarrow}$. Two automata G_1 and G_2 are *nonconflicting* if $G_1 \parallel G_2$ is nonblocking. ■

To enable reasoning about conflicts in a modular way, a notion of *conflict equivalence* of automata has been developed in [5]. The main results are recalled in the following. According to process-algebraic testing theory, two automata are considered as equivalent if they both respond in the same way to all tests of a certain type [7]. For *conflict equivalence*, a *test* is an arbitrary automaton, and the *response* is the observation whether the test composed with the automaton in question is nonblocking or not.

Definition 6: Two automata G_1 and G_2 are said to be *conflict equivalent*, $G_1 \simeq_{\text{conf}} G_2$, if, for any test T, $G_1 \parallel T$ is nonblocking if and only if $G_2 \parallel T$ is nonblocking. ■

The properties of conflict equivalence are studied in [5]: it suffices to consider deterministic tests in definition 6, and conflict equivalence is a congruence with respect to synchronous composition and hiding. Furthermore, the following algebraic characterisation is given.

Definition 7: Let G be an automaton with alphabet Σ and let $s \in \Sigma^*$. A language $\mathcal{C} \subseteq \Sigma^*\omega$ is called a *nonconflicting completion* for s in G, if for every test $T = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ such that G and T are nonconflicting and $Q^i \overset{s}{\Rightarrow} q$, there exists a completion $t \in \mathcal{C}$ such that $q \overset{t}{\Rightarrow}$. ■

Definition 8: The *nonconflicting completion semantics* of an automaton G is

$$\begin{aligned}\text{CC}(G) \ = \ \{\, (s, \mathcal{C}) \in \Sigma^* \times \mathbb{P}(\Sigma^*\omega) \mid \mathcal{C} \text{ is a nonconflicting}\\ \text{completion for } s \text{ in } G \,\}\ .\end{aligned}$$
■

The interpretation of a pair $(s, \mathcal{C})$ in the nonconflicting completion semantics is that a test accepting s *must* accept at least one of the completions in $\mathcal{C}$ after s to be nonconflicting with the automaton. The following result from [5] states that two automata are conflict equivalent if and only if their nonconflicting completion semantics coincide.

Theorem 1 (from [5]): Let G_1 and G_2 be two automata. Then $G_1 \simeq_{\text{conf}} G_2$ if and only if $\text{CC}(G_1) = \text{CC}(G_2)$. □

Given an automaton G, in general there is no unique minimal automaton that is conflict equivalent to G. For example, Fig. 1 shows three automata G_1, G_2, and G_3 that are all conflict equivalent and minimal, both in the number of states and in the number of transitions, but not isomorphic.

To see that these automata are conflict equivalent, note that $\text{CC}(G_1) = \text{CC}(G_2) = \text{CC}(G_3) = \{(\varepsilon, \alpha\alpha\alpha^*\omega)$, $(\alpha, \alpha\alpha^*\omega)$, $(\alpha\beta, \omega)$, $(\alpha\alpha\alpha^*, \alpha^*\omega)$, $(\alpha\alpha\beta^*, \beta^*\omega)\}$, if only the minimal nonconflicting completions are given and with a slight abuse of notation since the first element of every pair really should be a string. By Theorem 1, these three automata are all conflict equivalent.

To see that each of the three automata is minimal in the number of states, another result from [5] is needed: *for nonblocking automata*, conflict equivalence implies failures equivalence [6]. Therefore, any automaton that is conflict equivalent to G_1, e.g., must also be failures equivalent to G_1. The failure pairs of G_1 are $(\varepsilon, \{\beta, \omega\})$, $(\alpha, \{\beta, \omega\})$, $(\alpha\alpha\alpha^*, \{\beta\})$, $(\alpha\alpha\beta^*, \{\alpha\})$, and $(\alpha\beta, \{\alpha, \beta\})$, where only maximal refusals have been given—all subsets of these are

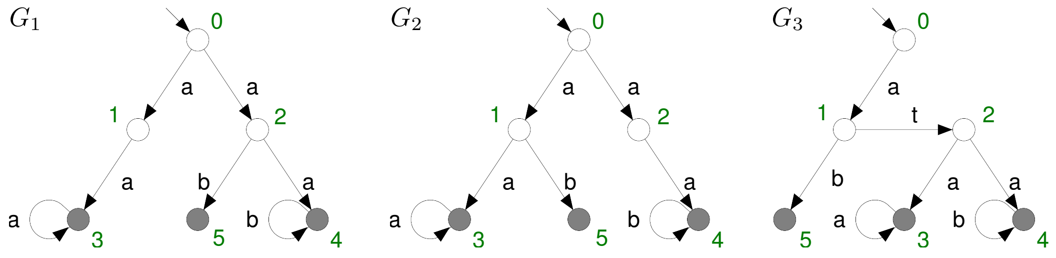

Fig. 1. Example of the nonuniqueness of minimal conflict equivalent systems.

also refusals. For two automata to be failures equivalent they must possess exactly the same failures. Then there must be at least one distinct state per maximal refusal set. By examining the failures, four different maximal refusal sets are found: $\{\beta,\omega\}$, $\{\beta\}$, $\{\alpha\}$, and $\{\alpha,\beta\}$. However, four states are not enough to produce the failures above. The existence of a refusal set for $\alpha\beta$ suggests that after accepting α, despite the refusal set $\{\beta,\omega\}$, the automaton must also have a state accepting β. This indicates that apart from a state refusing $\{\beta,\omega\}$, there must also be a state with maximal refusal set either $\emptyset$ or $\{\omega\}$. Moreover, if there is only one state representing the refusal set $\{\beta,\omega\}$, then there must be a selfloop on α in the initial state. This, however, would require $\{\beta,\omega\}$ to be a refusal set also of $\alpha\alpha$ which it is not. This proves that there must be at least six states in an automaton that is conflict equivalent to the automata in Fig. 1.

D. Equivalence Relations

An *equivalence relation* is a binary relation that is reflexive, symmetric and transitive. Given an equivalence relation $\sim$ on a set Q, the equivalence class of $q \in Q$ with respect to $\sim$, denoted $[q]$, is defined as $[q] = \{\, q' \in Q \mid q' \sim q \,\}$.

An equivalence relation on a set Q partitions Q into the set of its equivalence classes $Q/\!\sim\; = \{\, [q] \mid q \in Q \,\}$. For subsets Q' of Q, $Q'/\!\sim$ may cover more than Q'. The following definition is standard.

Definition 9: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ be an automaton and let $\sim\; \subseteq Q \times Q$ be an equivalence relation. The *abstraction* $G/\!\sim$ of G with with respect to $\sim$ is

$$G/\!\sim \;=\; \langle Q/\!\sim, \Sigma, \rightarrow/\!\sim, Q^i/\!\sim, Q^m/\!\sim \rangle \tag{5}$$

where $\rightarrow/\!\sim\; = \{\, [p] \xrightarrow{\sigma} [q] \mid p \xrightarrow{\sigma} q \,\}$. ■

It follows immediately from this definition that every path in the original automaton also occurs in its abstract version.

Lemma 1: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ be an automaton and let $\sim\; \subseteq Q \times Q$ be an equivalence relation. For every path

$$q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} q_n \tag{6}$$

in G it holds that

$$[q_0] \xrightarrow{\sigma_1} [q_1] \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} [q_n] \tag{7}$$

in $G/\!\sim$. □

Conversely, it cannot be guaranteed that a path in the abstract automaton also occurs in the original automaton. This is only the case if additional conditions are satisfied.

Definition 10: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ be an automaton. The binary relation $\sim_{\text{inc}} \subseteq Q \times Q$ is defined such that $q \sim_{\text{inc}} q'$ if

$$Q^i \overset{\varepsilon}{\Rightarrow} q \iff Q^i \overset{\varepsilon}{\Rightarrow} q' \;;$$
$$\forall p \in Q \;,\; \forall \sigma \in \Sigma : p \overset{\sigma}{\Rightarrow} q \iff p \overset{\sigma}{\Rightarrow} q' \;. \qquad ■$$

$\sim_{\text{inc}}$ can be shown to be an equivalence relation. This relation equates states that in some sense are always reached in the same way. This additional requirement suffices to establish a converse of Lemma 1.

Lemma 2: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ be an automaton, and let $\sim\; \subseteq Q \times Q$ be an equivalence relation such that $\sim\; \subseteq \;\sim_{\text{inc}}$. Then, for every path

$$\tilde{q}_0 \overset{\sigma_1}{\Rightarrow} \tilde{q}_1 \overset{\sigma_2}{\Rightarrow} \dots \overset{\sigma_n}{\Rightarrow} \tilde{q}_n \tag{8}$$

in $G/\!\sim$ there exist states $q_i \in \tilde{q}_i$, for $i = 0, \dots, n$, such that

$$q_0 \overset{\sigma_1}{\Rightarrow} q_1 \overset{\sigma_2}{\Rightarrow} \dots \overset{\sigma_n}{\Rightarrow} q_n \tag{9}$$

in G. □

The proof (by induction on n) is left out for lack of space.

III. Reduction Rules

To exploit the conflict equivalence concept for verification purposes, a method for obtaining a simplified, conflict equivalent representation of a given automaton is needed. No algorithm is known that, given a nondeterministic automaton, produces a smallest possible conflict equivalent automaton, neither with respect to the number of states nor transitions. Such minimality is not necessary for the purpose of verification. Instead it is preferred from a usable reduction algorithm to reduce the number of states and transitions as much as possible without requiring too much computational effort. This section discusses some possible methods.

A. Observation Equivalence

Observation equivalence, or *weak bisimulation* [9], is a strong equivalence of nondeterministic automata that considers only states with the same structure of future behaviour as equivalent.

Definition 11: An equivalence relation $\sim$ is said to be a *weak bisimulation* if $p \sim q$ implies that for any $s \in \Sigma^*$

$$\text{if } p \overset{s}{\Rightarrow} p' \text{ then } \exists q' \text{ such that } q \overset{s}{\Rightarrow} q' \text{ and } p' \sim q' \;;$$
$$\text{if } q \overset{s}{\Rightarrow} q' \text{ then } \exists p' \text{ such that } p \overset{s}{\Rightarrow} p' \text{ and } p' \sim q' \;.$$

Two automata G_1 and G_2 are observation equivalent if there exists a weak bisimulation $\sim$ such that for each initial state q_1

of G_1 there exists an initial state q_2 of G_2 such that $q_1 \sim q_2$, and vice versa. ■

It is established in [5] that observation equivalent automata are also conflict equivalent. Thus, it is possible to use existing algorithms [10]–[12] for observation equivalence straight away for minimisation with respect to conflict equivalence. This alone gives a significant reduction of the state space.

The authors' implementation for this reduction method is based on [11]. This algorithm is a generalised version of the algorithm in [10] and is claimed to have computational complexity $O(m \log n)$, where m is the number of transitions and n is the number of states. However, since the algorithm in [11] is designed for bisimulation equivalence, the automaton has to be augmented so that for all $\sigma \in \Sigma$, $p \overset{\sigma}{\Rightarrow} q$ always implies $p \xrightarrow{\sigma} q$. Thus m may be very large—in the order of n^2k where k is the number of events in the alphabet.

B. Conflict Equivalence Preserving Reductions

While algorithms for observation equivalence are well-established, they only achieve a limited amount of reduction. There are many cases where two automata are conflict equivalent but not observation equivalent. Therefore, the authors have worked out a set of *reduction rules* that can be used to reduce the state space of an automaton preserving conflict equivalence. These rules are applied locally to a few states of an automaton with particular structure and can be applied repeatedly to any matching parts of an automaton.

The following rules try to identify *conflict equivalent states* in an automaton, i.e., states that represent future behaviours that cannot be distinguished by conflict equivalence. Such conflict equivalent states can be merged without affecting possible conflicts with other components.

1) Active Events Rule: Two states that are equivalent with respect to $\sim_{\text{inc}}$ and have the same active events are conflict equivalent.

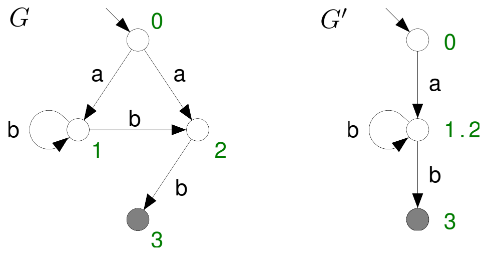

Fig. 2. Example of application of the Active Events Rule.

Example 1: 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 $\sim_{\text{inc}}$. Furthermore, they both have the same active event, β. Thus, the Active Events Rule can be applied, states 1 and 2 are conflict equivalent and can be collapsed into a single state 1.2 as shown in G'. ■

Theorem 2: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ be an automaton, and let $\sim \subseteq Q \times Q$ be an equivalence relation such that $\sim \subseteq \sim_{\text{inc}}$ and for all $p, q \in Q$ such that $p \sim q$ it holds that $\Sigma(p) = \Sigma(q)$. Then $G \simeq_{\text{conf}} G/\sim$. □

Proof: It suffices to show for any deterministic test T that, if $T \parallel G$ is nonblocking, so is $T \parallel G/\sim$, and vice versa.

($\Rightarrow$) Assume that $T \parallel G$ is nonblocking. Let $s \in \Sigma^*$ be a string accepted by T such that $Q^i/\sim \overset{s}{\Rightarrow} \tilde{q}$ in $G/\sim$. By Lemma 2, there exists $q \in Q$ such that $q \in \tilde{q}$ and $Q^i \overset{s}{\Rightarrow} q$ in G. Since $T \parallel G$ is nonblocking there exists $t \in \Sigma^*$ such that $st\omega$ is accepted by T and $q \overset{t\omega}{\Rightarrow}$ in G. By Lemma 1, this implies $\tilde{q} = [q] \overset{t\omega}{\Rightarrow}$ in $G/\sim$. Since this holds for all reachable states in $G/\sim$, it follows that $T \parallel G/\sim$ is nonblocking.

($\Leftarrow$) Assume $T \parallel G/\sim$ is nonblocking. Let $s \in \Sigma^*$ be a string accepted by T such that $Q^i \overset{s}{\Rightarrow} q$ in G. By Lemma 1, it holds that $Q^i/\sim \overset{s}{\Rightarrow} [q]$ in $G/\sim$. Then, since $T \parallel G/\sim$ is nonblocking, there exists $t \in \Sigma^*$ such that $[q] \overset{t\omega}{\Rightarrow}$ and $st\omega$ is accepted by T. By Lemma 2, there exists $q' \in [q]$ such that $q' \overset{t\omega}{\Rightarrow}$ in G. Note that $q \sim q'$ and therefore $\Sigma(q) = \Sigma(q')$. Thus, the first event of $t\omega$, say σ, is active in both q and q'. Also, σ is accepted by T after s since it is the first event of t. If $\sigma = \omega$, it follows that $q \overset{\omega}{\Rightarrow}$. Otherwise let $q \overset{\sigma}{\Rightarrow} p$. From Lemma 1 it follows that $[q] \overset{\sigma}{\Rightarrow} [p]$ in $G/\sim$. Now, since $T \parallel G/\sim$ is nonblocking there exists string $u \in \Sigma^*$ such that $[p] \overset{u\omega}{\Rightarrow}$ and $s\sigma u\omega$ is accepted by T. By Lemma 2, there exists $p' \in [p]$ such that $p' \overset{u\omega}{\Rightarrow}$ in G. Now, since $p \sim p'$, by (10), $q \overset{\sigma}{\Rightarrow} p$ implies that $q \overset{\sigma}{\Rightarrow} p'$, which means that there is a path $Q^i \overset{s}{\Rightarrow} q \overset{\sigma}{\Rightarrow} p' \overset{u\omega}{\Rightarrow}$ in G. Since this holds for all reachable states in G, it is clear that $T \parallel G$ is nonblocking. ■

2) Silent Continuation Rule: Two states that are equivalent with respect to $\sim_{\text{inc}}$ and from which *stable* states, i.e., states without outgoing τ transitions, can be reached via a *nonempty* sequence of τ transitions, are conflict equivalent.

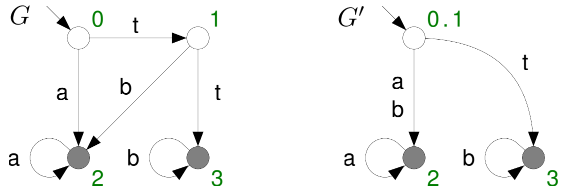

Fig. 3. Example of application of the Silent Continuation Rule.

Example 2: In Fig. 3, states 0 and 1 in G are both "initial" since they both can be reached silently from the initial state 0. This is enough to satisfy $\sim_{\text{inc}}$ 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 state 3, which has no outgoing τ transitions. Thus, by the Silent Continuation Rule, states 0 and 1 in G are conflict equivalent and can be collapsed into state 0.1 as shown in G'. ■

Theorem 3: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ be an automaton, and let $\sim \subseteq Q \times Q$ be an equivalence relation such that $\sim \subseteq \sim_{\text{inc}}$, and for all $p, q \in Q$ such that $p \sim q$ it holds that either $p = q$ or there exist stable states $p', q' \in Q$ such that $p \overset{\tau}{\Rightarrow} p'$ and $q \overset{\tau}{\Rightarrow} q'$. Then $G \simeq_{\text{conf}} G/\sim$. □

Proof: It suffices to show for any deterministic test T that, if $T \parallel G$ is nonblocking, so is $T \parallel G/\sim$, and vice versa.

($\Rightarrow$) Same as in the proof for Theorem 2.

($\Leftarrow$) Assume that $T \parallel G/\sim$ is nonblocking. Let $s \in \Sigma^*$ be a string accepted by T such that $Q^i \overset{s}{\Rightarrow} q$ in G. By Lemma 1 it holds that $Q^i/\sim \overset{s}{\Rightarrow} [q]$ in $G/\sim$. Consider two cases.

(1) $[q] = \{q\}$. Since $T \parallel G/\sim$ is nonblocking, from $[q]$ there exists a string $t \in \Sigma^*$ such that $st\omega$ is accepted by T

and $[q] \overset{t\omega}{\Rightarrow}$ in $G/\!\sim$. By Lemma 2 and since q is the only state in $[q]$, this means $q \overset{t\omega}{\Rightarrow}$ in G.

(2) $[q] \neq \{q\}$, i.e., there are at least two states $q_1, q_2 \in [q]$. Then $q_1 \sim q_2$ for $q_1 \neq q_2$, so for $q \in [q]$ there exists a stable state p such that $q \overset{\tau}{\Rightarrow} p$. By Lemma 1, this implies $[q] \overset{\tau}{\Rightarrow} [p]$. Since p has no outgoing τ-transitions, it is not $\sim$-equivalent to any other state and thus $[p] = \{p\}$. That is, from every case (2) situation, a case (1) situation can be reached. ■

3) Certain Conflicts Rule: In an automaton G, all states q such that $Q^i \xrightarrow{s} q$ and $(s, \emptyset) \in \mathrm{CC}(G)$ can be replaced by a single blocking state that has no outgoing transitions.

This method has been introduced in [8], where a proof for its soundness can be found. Strings $s \in \Sigma^*$ such that $(s, \emptyset) \in \mathrm{CC}(G)$ are called *certain conflicts* of G, because it is known that any test T running in parallel with G that can accept such a string s must be conflicting with G.

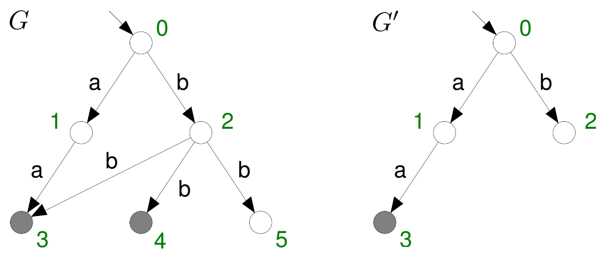

Fig. 4. Example of application of the Certain Conflicts Rule.

Example 3: In Fig. 4, since the string $\beta\beta$ may lead to the blocking state 5, any test that is to avoid blocking must avoid accepting $\beta\beta$. In this case, this implies that already after accepting just β, the automaton reaches a certain conflicts situation. Therefore $G \simeq_{\mathrm{conf}} G'$. ■

IV. Modular Nonblocking Verification

This section describes a method to check whether a large system of automata $G_1, G_2, \ldots, G_n$ is conflicting or not. Instead of building the synchronous product

$$G_1 \parallel G_2 \parallel \cdots \parallel G_n \ , \qquad (10)$$

conflict equivalent reductions can be used to replace an automaton G_i by a simpler version G_i'. As long as $G_i \simeq_{\mathrm{conf}} G_i'$ it readily follows from Def. 6 that the simplified system is nonblocking if and only if the original is. In this way, all automata for which conflict equivalent reductions are possible can be replaced before any synchronous product is computed. Since the state space of the synchronous product grows exponentially, even small savings in the number of states at this early stage can have considerable impact later.

When no more reductions are possible, the synchronous product is computed stepwise and with intermediate reductions. That is, first a subset of the automata in the system are replaced by their synchronous product H. Typically at this stage, there will be some set Υ of events that are *local*—used only by H and by no other automaton in the system. If so, these events can be hidden and the automaton $H \setminus \Upsilon$ replaces H. Finally, an attempt to simplify this intermediate result is made before composing a new set of automata.

Eventually, this procedure simplifies the system to a single automaton which does not share any events with other components. Therefore, all events can be hidden from this final automaton and it can be simplified in such a way that it is trivial to determine whether it is blocking or nonblocking. Due to the congruence results for conflict equivalence [5], the final automaton is guaranteed to be nonblocking if and only if the original system (10) is nonblocking. Clearly, if some intermediate automaton turns out to have no marked states then verification is also done—the end result cannot have any marked states either, so the system is blocking.

The above method leaves many choices for the order in which the automata are composed and simplified. It is not immediately clear how good performance can be achieved, but a reasonable strategy is to attempt to compose automata that will yield as many local events as possible. In some cases, the best strategy may be to construct small subsystems first, while in other cases it may be better to add automata one by one to a single growing subsystem.

For evaluation, a number of different heuristics have been examined. They all follow the same two-step approach to select which automata to compose. In the first step, some candidate sets of automata are computed to be used in the next step. This to avoid considering all combinations. The following heuristics are implemented for the first step.

minT. Candidates are all automata pairs containing the automaton with the fewest transitions.

maxS. Candidates are all automata pairs containing the automaton with the most states.

mustL. For each event there is a candidate which is the set of automata using that particular event.

The second step tries to identify the best candidate. The implemented heuristics attempt to keep the number of states after synchronous composition, hiding and simplification as small as possible. A prediction is made based on the assumption that a large number of shared (or *common*) events leads to a small synchronous product, and that a large number of local events (that can be hidden) is likely to yield many opportunities to apply the reduction procedures.

maxL. Choose the candidate with the highest proportion of local events.

maxC. Choose the candidate with the highest proportion of common events.

minS. Choose the candidate for which the product of the number of states in the automata is smallest.

In both steps, if one heuristic fails to distinguish two top candidates, another heuristic is used to make a decision. The order of priority in this regard is the order in which they are presented above per default, but with a particular chosen heuristic placed first in different experiments.

V. Experimental Results

The algorithm has been implemented in the DES software tool `Supremica` [?] and tested on a number of examples found in the literature. The examples include complex industrial models and case studies taken from various application areas such as manufacturing systems, communication protocols, and automotive body-electronics. In addition, a few very large examples with regular structure have been analysed. In the following, all examples considered are listed, followed by an evaluation of the experimental data.

A. Industrial Models

The following list shows the industrial examples analysed, together with the names of corresponding automata models, also referred to in Table I.

Car Central Locking System. A large DES model of the central locking system of a BMW car, taken from the KorSys project. `verriegel3`, `verriegel3b`, `verriegel4`, `verriegel4b`.

Automated Guided Vehicle Coordination. Based on the Petri net model in [13, page 119]. To make the example more interesting, there is also a variant with an additional zone added at the input station, which makes the system blocking. `AGV`, `AGVb`.

Production Cell for Mounting Frames. A model of a production cell in a metal-processing plant [14]. `fzelle`.

Flexible Production Cell. Another production cell, based on [15]. `ftechnik`, `ftechnik_nocoll`.

Intertwined Product Cycles. Two types of products are produced using two machines. The products must move between the machines twice and in opposite directions [16]. `IPC`.

Simple/Parallel Manufacturing System. Two highly modular models of manufacturing systems [17]. `SMS`, `PMS`.

Train testbed. A model of a toy railroad system [18]. `tbed`.

Car Window Control System A model of the behaviour of some parts of the electronics in a car [19]. `big_bmw`.

Atelier Inter-établissement de Productique. A very complex model of an automated manufacturing system [17]. `AIP`.

B. Parameterised Models

In addition, some models of more academic flavour have been analysed. They are instantiations of a number of identical subsystems connected to each other in a structured way. Due to their regular structure, these models can easily be scaled up to arbitrary size.

Since the subsystems are loosely connected, the state spaces of these models are enormous—truly exponential in the number of subsystems—whereas the “real” complexity of the problem is small since the problem essentially is the same for any number of instantiations. However, the verification algorithm does not exploit symmetry or the fact that it is solving a large number of identical problems: each subsystem is treated as if it was unique.

Dining Philosophers. This is the classical deadlocking (and blocking) example, here with a large number of philosophers. `256philo`, `512philo`, `1024philo`.

Arbiter. A model of a tree arbiter used to control access to a shared resource [20]. It can be parameterised for a varying number of users. `256arbiter`, `512arbiter`, `1024arbiter`.

Transfer line. Example from [21], with a parameterised number of serially connected cells. `128transfer`, `256transfer`, `512transfer`.

C. Evaluation

The results of all experiments can be found in Table I. The table shows the approximate size of the state space of the problems, the largest number of states encountered in a single automaton during verification and also the largest number of transitions, the time for the verification, and the result. Furthermore, the table shows the number of times the state space could be reduced (by one state) with the help of a certain rule. The “Other” column represents conflict equivalence rules not presented in this paper, and “OES” and “OET” represent observation equivalent reductions of states and transitions, respectively. The last columns tell which heuristics was used in the experiment. The heuristics **minT** and **maxL** have been used unless their performance was much worse than some other heuristic.

All experiments were conducted on a standard desktop PC with a 2.67 GHz processor and 512 MB of RAM. Most systems can be checked for nonblocking in a few seconds, without constructing automata with more than a few thousand states and transitions. For the parameterised models, the time and memory consumption grows linearly in the number of subsystems. These results clearly show that modular verification successfully avoids state-space explosion to a great extent.

For most examples, observation equivalence minimisation gives the most significant reduction. In particular, it is interesting to see that in the example `SMS` none of the conflict equivalence rules can be applied, while for the closely related example `PMS` the trend is the opposite. For examples that come in both a blocking and nonblocking variant, the blocking variant often verifies much faster. This is due to early termination when an intermediate result has no marked states.

Regarding the instantiated examples, the time consumption and the size of the intermediates suggests that the computational complexity is close to linear. The small intermediate models indicate that essentially the same problem is solved over and over a number of times that is linear in the number of automata. In the philosopher examples, the heuristic **minT**/**maxL** fails to find this symmetry which is why another strategy has been used. Presumably, a closer look at this example may reveal another reduction rule.

VI. Conclusions

A new method to check for large modular discrete event systems whether they are nonblocking has been presented. The method tries to reduce automata as much as possible before synchronous composition is computed. Experimental results show that the method can cope extremely well with very large industrial examples and brings considerable improvements over standard state-exploration algorithms.

TABLE I

EXPERIMENTAL RESULTS FOR NONBLOCKING VERIFICATION.

Example		**Size**	**Max**		**Time**	**Block**	**Reduction**						**Heuristic**	
Name	**Aut**		**States**	**Trans**			**AE**	**SC**	**CC**	**Other**	**OES**	**OET**	**Step 1**	**Step 2**
AGV	16	$2.6 \cdot 10^7$	413	1272	0.50s	false	39	19	0	131	91	1316	minT	maxL
AGVb	17	$2.3 \cdot 10^7$	255	778	0.35s	true	41	36	236	71	21	695	minT	maxL
verriegel3	53	$9.7 \cdot 10^8$	1239	5261	3.4s	false	88	80	0	115	581	2153	minT	maxL
verriegel3b	52	$1.3 \cdot 10^9$	11	46	0.12s	true	0	0	3	1	2	8	minT	maxL
verriegel4	65	$4.5 \cdot 10^{10}$	1897	8353	5.0s	false	130	125	0	201	963	3754	minT	maxL
verriegel4b	64	$6.3 \cdot 10^{10}$	11	46	0.13s	true	0	0	3	1	2	8	minT	maxL
big_bmw	31	$3.1 \cdot 10^7$	187	1474	0.91s	false	0	0	0	0	246	172	minT	maxL
SMS	11	312	17	22	0.24s	false	0	0	0	0	30	3	minT	maxL
PMS	38	$5.7 \cdot 10^6$	85	382	0.39s	false	18	8	0	63	6	470	minT	maxL
IPC	12	20592	256	985	0.41s	true	26	17	65	39	165	608	minT	maxL
ftechnik	36	$1.2 \cdot 10^8$	66	698	0.92s	true	4	11	78	2	173	117	minT	maxL
ftechnik_nocoll	42	$1.2 \cdot 10^8$	135	2434	0.35s	true	10	15	129	4	97	91	mustL	minS
tbed	84	13680	1800	9313	12s	true	0	6	2219	43	1018	457	mustL	minS
fzelle	67	$3.0 \cdot 10^{11}$	810	3169	5.4s	false	106	176	4	280	847	3500	mustL	minS
AIP	117	$1.0 \cdot 10^9$	1050	4200	6.2s	false	135	150	10	159	1090	4296	mustL	minS
256philo	512	$5.4 \cdot 10^{168}$	35	86	6.6s	true	512	1	0	1535	3548	9769	maxS	maxL
512philo	1024	$2.9 \cdot 10^{337}$	35	86	17s	true	1024	1	0	3071	7132	19599	maxS	maxL
1024philo	2048	$8.5 \cdot 10^{674}$	35	86	41s	true	2048	1	0	6143	14300	39260	maxS	maxL
128transfer	640	$1.6 \cdot 10^{231}$	24	67	5.7s	false	381	3	0	894	254	2055	minT	maxL
256transfer	1280	$2.4 \cdot 10^{462}$	24	67	29s	false	765	3	0	1790	510	4103	minT	maxL
512transfer	2560	$5.8 \cdot 10^{924}$	24	67	3.3m	false	1533	3	0	3582	1022	8199	minT	maxL
128arbiter	637	$3.9 \cdot 10^{158}$	318	941	14s	false	1205	1273	1482	1478	6226	6985	minT	maxL
256arbiter	1277	$3.8 \cdot 10^{684}$	318	941	32s	false	2420	2593	3083	2987	13097	14243	minT	maxL
512arbiter	2557	$5.7 \cdot 10^{4539}$	318	941	69s	false	4848	5217	6211	5998	26581	28671	minT	maxL

There are several ways that the modular algorithm can be improved. Different heuristics and alternative reduction methods need to be studied. The result of the simplification also depends on the order in which the reduction rules are applied. In all the experiments conducted this order has been the same but clearly this is something to consider for further improving the performance of the algorithm.

Moreover, it is conceivable to use a coarser equivalence: the very general conflict equivalence used here considers *all possible contexts*, better reduction is possible if more aspects of *the current context* of the system under verification can be taken into account.

REFERENCES

[1] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[2] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer, Sept. 1999.

[3] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proc. of the 15th Triennial World Congress of the International Federation of Automatic Control*, Barcelona, Spain, July 2002.

[4] B. A. Brandin, R. Malik, and P. Malik, "Incremental verification and synthesis of discrete-event systems guided by counter-examples," *IEEE Trans. Contr. Syst. Technol.*, vol. 12, no. 3, pp. 387–401, May 2004.

[5] R. Malik, D. Streader, and S. Reeves, "Fair testing revisited: A process-algebraic characterisation of conflicts," in *Proc. 2nd Int. Symp. Automated Technology for Verification and Analysis, ATVA 2004*, ser. LNCS, F. Wang, Ed., vol. 3299. Taipei, Taiwan: Springer, 2004, pp. 120–134.

[6] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.

[7] R. D. Nicola and M. C. B. Hennessy, "Testing equivalences for processes," *Theoretical Comput. Sci.*, vol. 34, no. 1–2, pp. 83–133, 1984.

[8] R. Malik, "On the set of certain conflicts of a given language," in *Proc. 7th Int. Workshop on Discrete Event Systems, WODES '04*, Reims, France, Sept. 2004, pp. 277–282.

[9] R. Milner, *Communication and Concurrency*. Prentice-Hall, 1989.

[10] R. Paige and R. E. Tarjan, "Three partition refinement algorithms," *SIAM J. Comput.*, vol. 16, no. 6, pp. 973–989, 1987.

[11] S. Westin, "Fast decision of strong bisimulation equivalence using partition refinement," Licentiate thesis, Dept. of Computer Sciences, Chalmers University of Technology, 1989.

[12] J. Eloranta, "Minimizing the number of transitions with respect to observation equivalence," *BIT*, vol. 31, no. 4, pp. 576–590, 1991.

[13] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, 1998.

[14] C. Lewerentz and T. Lindner, *Formal Development of Reactive Systems—Case Study Production Cell*, ser. LNCS. Springer, 1995, vol. 891, pp. 7–19.

[15] A. Lötzbeyer and R. Mühlfeld, "Task description of a flexible production cell with real time properties," FZI, Karlsruhe, Germany, Tech. Rep., 1996. [Online]. Available: http://www.fzi.de/divisions/prost/projects/korsys/korsys.html

[16] F. Lin and W. M. Wonham, "Decentralized control and coordination of discrete-event systems with partial observation," *IEEE Trans. Automat. Contr.*, vol. 35, no. 12, pp. 1330–1337, Dec. 1990.

[17] R. J. Leduc, "Hierarchical interface-based supervisory control," Ph.D. dissertation, Dept. of Electrical and Computer Engineering, University of Toronto, Canada, 2002.

[18] ——, "PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective," Master's thesis, University of Toronto, Dept. of Computer and Electrical Engineering, Toronto, Canada, 1996.

[19] P. Malik, "From supervisory control to nonblocking controllers for discrete event systems," Ph.D. dissertation, Dept. of Computer Science, University of Kaiserslautern, Kaiserslautern, 2003.

[20] E. M. Clarke, D. E. Long, and K. L. McMillan, "Compositional model checking," in *Proceedings of the Fourth Annual Symposium on Logic in computer science*. Piscataway, NJ, USA: IEEE Press, 1989, pp. 353–362.

[21] W. M. Wonham, "Notes on control of discrete event systems," Dept. of Electrical and Computer Engineering, University of Toronto, Tech. Rep., 1999.