International Journal of Foundations of Computer Science
© World Scientific Publishing Company

CONFLICTS AND FAIR TESTING

ROBI MALIK DAVID STREADER STEVE REEVES
Department of Computer Science, University of Waikato
Hamilton, New Zealand

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

This paper studies *conflicts* from a process-algebraic point of view and shows how they are related to the testing theory of *fair testing.* Conflicts have been introduced in the context of discrete event systems, where two concurrent systems are said to be in conflict if they can get trapped in a situation where they are waiting or running endlessly, forever unable to complete their common task. In order to analyse complex discrete event systems, conflict-preserving notions of refinement and equivalence are needed. This paper characterises an appropriate refinement, called the *conflict preorder*, and provides a denotational semantics for it. Its relationship to other known process preorders is explored, and it is shown to generalise the fair testing preorder in process-algebra for reasoning about conflicts in discrete event systems.

Keywords: Process-algebra, discrete event systems, model checking, nonblocking.

1. Introduction

Conflicts are a common fault in the design of concurrent programs that can be very subtle and hard to detect [10, 26]. They have long been studied in the field of *discrete event systems* [6,23], which is applied to the modelling of complex, safety-critical systems [2,15,16]. In order to improve the reliability of such systems, techniques are needed to facilitate the design of systems that are free from conflict.

Two processes are said to be in conflict if they can reach a state from which no terminal state can be reached anymore. This includes both the possibility of *deadlock*, where processes are stuck and unable to continue at all, and *livelock*, where processes continue to run forever without achieving any further progress.

A discrete event system is free from conflict if, from every reachable state, all involved processes *can* cooperatively reach a terminal state together. It is not required that a terminal state is necessarily reached on every possible execution path, only that there exists an execution path to a terminal state. This corresponds to termination under the *strong fairness* assumption that every transition that is enabled infinitely often is taken eventually [1].

Such a concept of possible termination with respect to fairness assumptions is very useful for several applications, e.g., for communication protocols [21]. To be free from conflict is the underlying liveness assumption used for synthesis in supervisory control theory [6, 23]. To check for conflicts also is useful for fault detection in *model checking* [7], because a system that contains conflicts clearly cannot be guaranteed to terminate.

Unfortunately, it is very difficult to reason about conflicts in a modular way. If two components are free from conflict individually, they may well be involved in a conflict when running together, and vice versa [26]. This makes it impossible to apply most methods of abstraction common in model checking [7, 8] when trying to verify systems to be free from conflict.

Several approaches of modelling and refinement have been proposed for discrete event systems [14, 26, 27], each suggesting a special way of designing systems such that they are free from conflict by construction. While giving some insight into how systems can be designed to be free from conflict, these techniques rely on interfaces provided by users and therefore cannot be applied automatically.

Research in process algebra has focused on general ways of composing systems, and identifying notions of refinement and equivalence that preserve properties of interest [25]. These ideas provide a very powerful abstract framework for modular reasoning about concurrent systems. However, standard process-algebraic approaches are either based on *failures* [12, 24], which make no fairness assumption and therefore cannot be applied for conflict analysis, or on *bisimulation* [20], where fairness is implicit in the observational semantics, but which makes unnecessarily fine distinctions.

The less common process-algebraic theory of *fair testing*, which has been developed independently by two groups of researchers in [3, 4] and [21], provides the formal framework needed to characterise conflict-preserving refinements. The present work applies and extends these results to conflicts in discrete event systems. To the best of our knowledge, it is the first to define a pre-congruence based on the elegant concept of hiding in process-algebra that also respects conflicts, and hence can be used for modular conflict analysis.

This paper is an extended version of [18], with more detailed proofs, and an elaborate discussion of the relationships between conflicts and fair testing in section 3.6. Section 2 introduces notations and provides a definition of conflicts. Section 3 presents the conflict preorder and explores its relationship to fair testing and other known semantics. The conflict preorder is shown to be the best possible refinement to reason about conflicts. Finally, section 4 contains some concluding remarks.

2. Notation

This section introduces the notations used throughout this paper. Processes are represented as labelled transition systems, with the possibility of nondeterminism which naturally arises from abstraction and hiding operations [12, 20, 24]. Process behaviour is described using languages, with notations taken from the background of discrete event systems theory [6, 23].

2.1. Languages

Traces and languages are a simple means to describe process behaviours. Their basic building blocks are *actions*, which are taken from a finite *alphabet* $\mathbf{A}$. Then $\mathbf{A}^*$ denotes the set of all finite *strings* or *traces* of the form $\alpha_1\alpha_2\cdots\alpha_k$ of actions from $\mathbf{A}$, including the *empty trace* ε. A *language* over $\mathbf{A}$ is any subset $\mathcal{L} \subseteq \mathbf{A}^*$.

The *catenation* of two traces $s, t \in \mathbf{A}^*$ is written as st. Strings and languages can also be catenated, e.g., $s\mathcal{L} \stackrel{\text{def}}{=} \{\, st \mid t \in \mathcal{L} \,\}$. The *prefix-closure* $\overline{\mathcal{L}}$ of a language $\mathcal{L} \subseteq \mathbf{A}^*$ is the set of all prefixes of traces in $\mathcal{L}$, i.e., $\overline{\mathcal{L}} \stackrel{\text{def}}{=} \{\, s \in \mathbf{A}^* \mid st \in \mathcal{L} \text{ for some } t \in \mathbf{A}^* \,\}$. A language $\mathcal{L}$ is called *prefix-closed* if $\mathcal{L} = \overline{\mathcal{L}}$.

The special action $\omega \notin \mathbf{A}$ is used to indicate successful termination. Then, $\mathbf{A}_\omega \stackrel{\text{def}}{=} \mathbf{A} \cup \{\omega\}$ denotes the set of actions including ω. The termination action only makes sense at the end of an execution, therefore all action sequences should belong to $\mathbf{A}^{*\omega} \stackrel{\text{def}}{=} \mathbf{A}^* \cup \mathbf{A}^*\omega$.

2.2. Processes

In the context of this paper, processes are modelled as nondeterministic *labelled transition systems*

$$P \;=\; (\mathbf{A}, Q, \rightarrow, q^\circ, Q^\omega)\ , \tag{1}$$

where $\mathbf{A}$ is the alphabet of actions, Q is the non-empty set of *states*, $\rightarrow\, \subseteq Q \times \mathbf{A} \times Q$ is the *transition relation*, $q^\circ \in Q$ is the *initial state*, and $Q^\omega \subseteq Q$ is the set of *success states*. The transition relation is also written as

$$q \xrightarrow{\alpha} q' \quad \text{if and only if} \quad (q, \alpha, q') \in\, \rightarrow\ . \tag{2}$$

If the transition relation can be described as a function, i.e., if $q \xrightarrow{\alpha} q_1$ and $q \xrightarrow{\alpha} q_2$ always implies $q_1 = q_2$, then P is called *deterministic*.

Labelled transition systems, i.e., processes, are represented graphically as shown in figure 1: states are represented as nodes, with the initial state highlighted by a thick border and success states shaded grey. The transition relation is represented by labelled edges.

Processes use the set Q^ω of success states to indicate the possibility of successful termination. In order to translate this into an action-based representation, every process is assumed to have a terminal state $\perp \in Q \setminus Q^\omega$, from which there are no outgoing transitions. Then the transition relation is extended to a relation $\rightarrow\, \subseteq Q \times \mathbf{A}_\omega \times Q$ by adding transitions

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

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

The action-labelled transition relation $\rightarrow$ is further extended to a string-labelled transition relation $\Rightarrow\, \subseteq Q \times \mathbf{A}^{*\omega} \times Q$ defined as

$$q \stackrel{\varepsilon}{\Rightarrow} q \qquad \text{for each} \qquad q \in Q\ ; \tag{4}$$

$$q \stackrel{s\alpha}{\Rightarrow} q' \qquad \text{if and only if} \qquad q \stackrel{s}{\Rightarrow} q_s \xrightarrow{\alpha} q' \text{ for some } q_s \in Q\ . \tag{5}$$

The set of all labelled transitions systems, and thus all processes, with action alphabet $\mathbf{A}$ is denoted by $\Pi_{\mathbf{A}}$. The transition relation is also defined for processes, denoting by

$$P \overset{s}{\Rightarrow} P' \tag{6}$$

that process $P \in \Pi_{\mathbf{A}}$ evolves into $P' \in \Pi_{\mathbf{A}}$ by executing actions $s \in \mathbf{A}^{*\omega}$. This is defined as $(\mathbf{A}, Q, \rightarrow, q^\circ, Q^\omega) \overset{s}{\Rightarrow} (\mathbf{A}, Q, \rightarrow, q', Q^\omega)$ for each $q^\circ \overset{s}{\Rightarrow} q'$. The notation $P \overset{s}{\Rightarrow}$ states that $P \overset{s}{\Rightarrow} P'$ for some $P' \in \Pi_{\mathbf{A}}$.

The possible behaviours of a process are defined by the set of action sequences or *traces* it can execute. The *language* $\mathcal{L}(P)$ and the *success language* $\mathcal{M}(P)$ of $P \in \Pi_{\mathbf{A}}$ are defined as

$$\mathcal{L}(P) \stackrel{\text{def}}{=} \{\, s \in \mathbf{A}^{*\omega} \mid P \overset{s}{\Rightarrow} \,\} \quad \text{and} \quad \mathcal{M}(P) \stackrel{\text{def}}{=} \{\, s \in \mathbf{A}^{*}\omega \mid P \overset{s}{\Rightarrow} \,\}\ . \tag{7}$$

$\mathcal{L}(P)$ contains all complete or incomplete traces that can be executed by a process, including or not including the termination action ω. This is a prefix-closed language. In contrast, $\mathcal{M}(P)$ contains only traces ending with ω, i.e., only those traces that lead to successful termination.

2.3. Synchronous Product

When several processes are running in parallel, lock-step synchronisation in the style of [12] is used. The synchronous product $P_1 \parallel P_2$ of two processes $P_1 = (\mathbf{A}, Q_1, \rightarrow_1, q_1^\circ, Q_1^\omega)$ and $P_2 = (\mathbf{A}, Q_2, \rightarrow_2, q_2^\circ, Q_2^\omega)$, both using the action alphabet $\mathbf{A}$, is defined as

$$P_1 \parallel P_2 \stackrel{\text{def}}{=} (\mathbf{A}, Q_1 \times Q_2, \rightarrow, (q_1^\circ, q_2^\circ), Q_1^\omega \times Q_2^\omega)\ , \tag{8}$$

where $(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')$ if and only if $q_1 \xrightarrow{\alpha}_1 q_1'$ and $q_2 \xrightarrow{\alpha}_2 q_2'$. Synchronisation is performed on all actions, including the termination action ω. Two processes can only terminate together when both are ready to terminate.

2.4. Conflicts

Given a process $P \in \Pi_{\mathbf{A}}$, it is desirable that every trace in $\mathcal{L}(P)$ can be completed to a trace in $\mathcal{M}(P)$, otherwise P may become unable to terminate. In discrete event systems theory, a process that may become unable to terminate is called *blocking*. This concept becomes more interesting when several processes are running in parallel—in this case the term *conflicting* is used instead. The following extends the standard definitions [23] to the case of nondeterministic processes.

Definition 1 A process $P \in \Pi_{\mathbf{A}}$ is said to be *nonblocking*, if for every trace $s \in \mathbf{A}^*$ and every $P' \in \Pi_{\mathbf{A}}$ such that $P \overset{s}{\Rightarrow} P'$ there exists a continuation $t \in \mathbf{A}^*\omega$ such that $P' \overset{t}{\Rightarrow}$. Otherwise P is said to be *blocking*.

Definition 2 Two processes $P_1, P_2 \in \Pi_{\mathbf{A}}$ are said to be *nonconflicting* if $P_1 \parallel P_2$ is nonblocking. Otherwise they are said to be *conflicting*.

In order to be nonblocking, or nonconflicting, it is sufficient that a terminal state *can* be reached in every possible situation. This is equivalent to termination

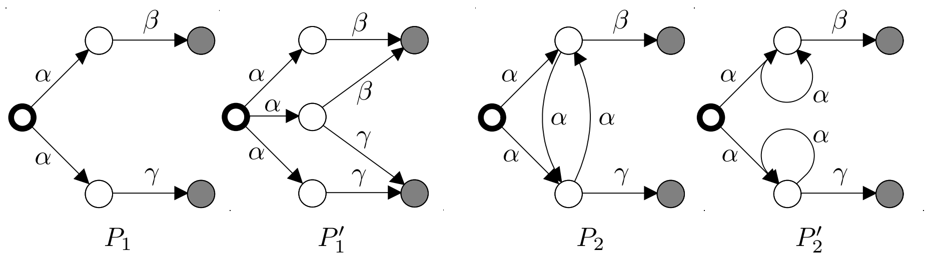

Fig. 1. Example processes.

under an implicit *strong fairness* assumption stating that "whenever a transition can occur infinitely often, it occurs infinitely often" [1]. Process P_2 in figure 1, e.g., is nonblocking, although it can theoretically execute an infinite sequence of α actions without ever terminating. Yet, since the β and γ transitions are enabled infinitely often in such an execution, strong fairness requires one of them to be taken eventually.

3. The Conflict Preorder

3.1. Finding Conflicts in Large Systems

An aim of this paper is to provide the framework for algorithms to determine whether a large system of concurrent processes is conflicting or not. The straightforward approach to check whether processes $P_1, P_2, \ldots, P_n$ are conflicting is to construct their synchronous product

$$P_1 \parallel P_2 \parallel \cdots \parallel P_n \tag{9}$$

and check whether it is blocking. This is done by checking whether a terminal state can be reached from every reachable state. Using symbolic representations such as BDDs [5] or IDDs [28], this approach has been used to analyse very large models. Yet, the technique always remains limited by the amount of memory available to store representations of the synchronous product.

An alternative approach avoids building the entire synchronous product by analysing smaller subsystems first. Modular reasoning can make it possible to replace the process P_1, e.g., by a simpler version P_1', and analyse the simpler system

$$P_1' \parallel P_2 \parallel \cdots \parallel P_n \; . \tag{10}$$

Such modular reasoning requires an appropriate notion of process equivalence. For the sake of conflict analysis, processes P_1 and P_1' in figure 1 are equivalent, since any other process T that is nonconflicting with either P_1 and P_1' must be able to execute α and then be able to continue both with β and with γ. On the other hand, processes P_2 and P_2' in figure 1 are not equivalent. A process T that can only execute any number of α actions followed by β is nonconflicting with P_2 but not with P_2'.

Such a notion of equivalence can be obtained using the process-algebraic framework of testing [11, 22] by considering conflicts as a testing paradigm. Section 3.2 formally introduces the notion of conflict equivalence. Section 3.3 discusses its congruence properties and shows that it is the best possible equivalence to reason about conflicts. Section 3.4 provides a denotational characterisation. Section 3.5 introduces the notion of *certain conflicts* as a characteristic property of conflict equivalence. Finally, sections 3.6 and 3.7 compare conflict equivalence with other process semantics.

3.2. Using Conflicts as a Testing Paradigm

The traditional testing framework [11, 22] defines preorders and equivalences that relate processes based on their responses to tests. This setting can be applied to obtain an equivalence based on conflicts: a *test* can be any process, and the test's *response* is the observation whether the test is conflicting with the given process or not. Then two processes are considered as equivalent if the responses of all tests are equal.

Definition 3 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$.

- P_1 is *less conflicting* than P_2, written $P_1 \lesssim_{\text{conf}} P_2$, if for every test $T \in \Pi_{\mathbf{A}}$, if P_2 and T are nonconflicting, then P_1 and T are also nonconflicting.
- P_1 and P_2 are *conflict equivalent*, written $P_1 \simeq_{\text{conf}} P_2$, if $P_1 \lesssim_{\text{conf}} P_2$ and $P_2 \lesssim_{\text{conf}} P_1$.

For example, processes P_1 and P_1' in figure 1 are conflict equivalent, while P_2 and P_2' are not.

Given a preorder based on tests such as $\lesssim_{\text{conf}}$, it is of interest [11] to reduce the set of tests needed to establish $P_1 \lesssim_{\text{conf}} P_2$. The following proposition shows that it is sufficient to consider deterministic tests only.

Definition 4 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$. Write $P_1 \lesssim^{\text{det}}_{\text{conf}} P_2$, if for every *deterministic* test $T \in \Pi_{\mathbf{A}}$, if P_2 and T are nonconflicting, then P_1 and T are also nonconflicting.

Proposition 1 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$. If $P_1 \lesssim^{\text{det}}_{\text{conf}} P_2$ then $P_1 \lesssim_{\text{conf}} P_2$.

Proof (sketch). Let $P_1 \lesssim^{\text{det}}_{\text{conf}} P_2$, and assume that $P_1 \lesssim_{\text{conf}} P_2$ does not hold. Then there exists a test $T \in \Pi_{\mathbf{A}}$ such that P_2 and T are nonconflicting, but P_1 and T are conflicting. Since P_1 and T are conflicting, there exists a trace $s \in \mathbf{A}^*$ such that $P_1 \,\|\, T \overset{s}{\Rightarrow} P_1' \,\|\, T'$, but there does not exist any $t \in \mathbf{A}^*\omega$ such that $P_1' \,\|\, T' \overset{t}{\Rightarrow}$. Then construct a deterministic process $T^{\text{det}} \in \Pi_{\mathbf{A}}$ such that

$$\mathcal{L}(T^{\text{det}}) = (\mathcal{L}(T) \setminus s\mathbf{A}^*) \cup s\mathcal{L}(T') \; ; \tag{11}$$

$$\mathcal{M}(T^{\text{det}}) = (\mathcal{M}(T) \setminus s\mathbf{A}^*) \cup s\mathcal{M}(T') \; . \tag{12}$$

A nondeterministic acceptor of these languages can be constructed from T by removing behaviours that extend s but are not possible in T'; then subset construction [13] is used to make it deterministic and obtain T^{det}. It can be proven that P_2 and T^{det} are nonconflicting, since P_2 and T are, but P_1 and T^{det} are conflicting. This contradicts the assumption $P_1 \lesssim^{\text{det}}_{\text{conf}} P_2$. $\square$

3.3. Congruence Properties

When studying process preorders such as $\lesssim_{\text{conf}}$, it is an important question how these preorders behave when processes are modified or combined by standard operations. It is desirable that relationships between two processes are preserved when the same operation is applied to both processes. A preorder that satisfies this condition is called a *pre-congruence.*

The congruence properties of fair testing have been studied in [3]. All these results can be extended for the conflict preorder. Below is a proof for the congruence result with respect to synchronous composition, which is simple in the terminology of conflicts.

Definition 5 Let $\lesssim \subseteq \Pi_{\mathbf{A}} \times \Pi_{\mathbf{A}}$ be a preorder on the set of processes.

- $\lesssim$ is a *pre-congruence* with respect to $\|$ if, for all processes $P_1, P_2, T \in \Pi_{\mathbf{A}}$ such that $P_1 \lesssim P_2$, it follows that $P_1 \| T \lesssim P_2 \| T$.
- $\lesssim$ *respects blocking* if, for all processes $P_1, P_2 \in \Pi_{\mathbf{A}}$ such that $P_1 \lesssim P_2$, if P_2 is nonblocking then P_1 also is nonblocking.

Proposition 2 $\lesssim_{\text{conf}}$ is a pre-congruence with respect to $\|$.

Proof. Let $P_1 \lesssim_{\text{conf}} P_2$ and $T \in \Pi_{\mathbf{A}}$. To see that $P_1 \| T \lesssim_{\text{conf}} P_2 \| T$, let $T' \in \Pi_{\mathbf{A}}$ be a test such that $P_2 \| T$ and T' are nonconflicting. Then $P_2 \| T \| T'$ is nonblocking or, equivalently, P_2 and $T \| T'$ are nonconflicting. Since $P_1 \lesssim_{\text{conf}} P_2$, it follows that P_1 and $T \| T'$ are nonconflicting, i.e., $P_1 \| T \| T'$ is nonblocking, i.e., $P_1 \| T$ and T' are nonconflicting. Therefore, $P_1 \| T \lesssim_{\text{conf}} P_2 \| T$. □

The conflict preorder can furthermore be shown to be a pre-congruence with respect to other process-algebraic operators, including generalised forms of parallel composition, hiding, prefixing, and renaming [3]. To have a pre-congruence with respect to hiding [24] is particularly important, because it means that this standard method of abstraction can also be used to simplify processes in a conflict-preserving way.

The conflict preorder can be characterised as the coarsest pre-congruence with respect to synchronous composition that respects blocking. In other words, any process equality that distinguishes processes according to their blocking behaviour and preserves synchronous composition is contained in conflict equivalence. This means that the conflict preorder is the best possible process refinement for reasoning about conflicts.

Proposition 3 $\lesssim_{\text{conf}}$ respects blocking.

Proof. Note that there exists a process $U_{\mathbf{A}} \in \Pi_{\mathbf{A}}$ such that $P \| U_{\mathbf{A}} = P$ for every $P \in \Pi_{\mathbf{A}}$. Let $P_1 \lesssim_{\text{conf}} P_2$, and let P_2 be nonblocking. Then P_2 and $U_{\mathbf{A}}$ are nonconflicting. Since $P_1 \lesssim_{\text{conf}} P_2$, it follows that P_1 and $U_{\mathbf{A}}$ are nonconflicting, i.e., $P_1 \| U_{\mathbf{A}} = P_1$ is nonblocking. □

Proposition 4 Let $\lesssim \subseteq \Pi_{\mathbf{A}} \times \Pi_{\mathbf{A}}$ be a pre-congruence with respect to $\|$ which respects blocking. Then $P_1 \lesssim P_2$ implies $P_1 \lesssim_{\text{conf}} P_2$.

Proof. Let $P_1 \lesssim P_2$, and let $T \in \Pi_{\mathbf{A}}$ such that P_2 and T are nonconflicting. Then $P_2 \,\|\, T$ is nonblocking. Furthermore, $P_1 \,\|\, T \lesssim P_2 \,\|\, T$ since $\lesssim$ is a pre-congruence with respect to $\|$. Since $\lesssim$ respects blocking it follows that $P_1 \,\|\, T$ is nonblocking, i.e., P_1 and T are nonconflicting. Hence $P_1 \lesssim_{\mathrm{conf}} P_2$. □

3.4. The Nonconflicting Completion Semantics

In addition to the definition of a preorder by referring to arbitrary test environments, it is desirable to have a denotational characterisation based on the structural properties of a process. Such characterisations have been introduced for fair testing in [3, 4, 21], but they are complicated and hard to relate to traditional characterisations such as failures [12, 25]. This section proposes an elegant characterisation of the conflict preorder, called the nonconflicting completion semantics, which has an intuitive interpretation based on the idea of conflicts.

The idea of the nonconflicting completion semantics is to list for each process P a set of conditions that have to be satisfied by a test T that is to be nonconflicting with P. Assume that such a test accepts a trace s that is also accepted by P. In order to be nonconflicting with P, such a test will be required to be able to terminate in certain ways. Each set of possibilities is combined as a language of possible completions, called a *nonconflicting completion* for s in P.

Definition 6 Let $P \in \Pi_{\mathbf{A}}$ be a process, and let $s \in \mathbf{A}^*$. A language $\mathcal{C} \subseteq \mathbf{A}^*\omega$ is called a *nonconflicting completion* for s in P, if for every test $T \in \Pi_{\mathbf{A}}$ such that P and T are nonconflicting and $T \overset{s}{\Rightarrow} T'$, there exists a completion $t \in \mathcal{C}$ such that $T' \overset{t}{\Rightarrow}$.

Definition 7 The *nonconflicting completion semantics* of $P \in \Pi_{\mathbf{A}}$ is

$$\begin{aligned} \mathrm{CC}(P) \overset{\text{def}}{=} \{\, & (s, \mathcal{C}) \in \mathbf{A}^* \times \mathbb{P}(\mathbf{A}^*\omega) \mid \\ & \mathcal{C} \text{ is a nonconflicting completion for } s \text{ in } P \,\} \ . \end{aligned} \tag{13}$$

This construction is similar in style to failures semantics [12]. But here, each trace is paired with sets of successful completions, namely with all its nonconflicting completions. Each pair $(s, \mathcal{C}) \in \mathrm{CC}(P)$ describes a condition that needs to be satisfied by a test that is to be nonconflicting with P. Such a test, if it can accept s, must afterwards always be able to terminate with at least one of the traces in $\mathcal{C}$.

Example 1 Consider process P_1 in figure 1, and assume that a test T running in parallel with P_1 accepts action α. In order to be nonconflicting with P_1, the test T, after having executed α, must be able to accept β and then terminate. This is because, after execution of α, process P_1 may be in a state from which the only possibility to terminate is β. Therefore, the nonconflicting completion semantics $\mathrm{CC}(P_1)$ of P_1 contains the pair $(\alpha, \{\beta\omega\})$. For similar reasons, it contains the pair $(\alpha, \{\gamma\omega\})$.

Similar conditions are associated with the other traces executed by P_1. Overall, the nonconflicting completion semantics of P_1 can be characterised by the following set.

$$\{(\varepsilon, \{\alpha\beta\omega, \alpha\gamma\omega\}), (\alpha, \{\beta\omega\}), (\alpha, \{\gamma\omega\}), (\alpha\beta, \{\omega\}), (\alpha\gamma, \{\omega\})\} \tag{14}$$

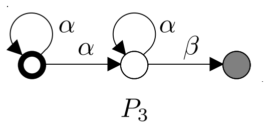

Fig. 2. Example with a not well-founded set of nonconflicting completions.

Yet, $\mathrm{CC}(P_1)$ is not equal to the above set. The above set only lists *minimal* nonconflicting completions. All supersets of these minimal nonconflicting completions are also nonconflicting completions and provide additional pairs to be included in $\mathrm{CC}(P_1)$

Example 2 Consider process P_2 in figure 1, and let T be a test accepting α. To be nonconflicting with P_2, T must be able to terminate after at least one of the traces $\beta, \alpha\gamma, \alpha\alpha\beta, \ldots$, and at least one of the traces $\gamma, \alpha\beta, \alpha\alpha\gamma, \ldots$ Therefore, $\mathrm{CC}(P_2)$ contains $(\alpha, \{(\alpha\alpha)^*\beta\omega, (\alpha\alpha)^*\alpha\gamma\omega\})$ and $(\alpha, \{(\alpha\alpha)^*\gamma\omega, (\alpha\alpha)^*\alpha\beta\omega\})$.

Consider a process $P \in \Pi_{\mathbf{A}}$ such that $P \overset{s}{\Rightarrow} P'$. Then any test T that is nonconflicting with P, after having executed s, must be able to terminate together with P'. Thus the set of all possible ways in which P' can terminate, i.e., the success language of P', forms a nonconflicting completion for s in P. This result is easy to prove.

Lemma 5 Let $P, P' \in \Pi_{\mathbf{A}}$ and $s \in \mathbf{A}^*$ such that $P \overset{s}{\Rightarrow} P'$. Then

$$(s, \mathcal{M}(P')) \in \mathrm{CC}(P) . \tag{15}$$

Proof. It is to be proven that $\mathcal{M}(P')$ is a nonconflicting completion for s in P. Let $T \in \Pi_{\mathbf{A}}$ be nonconflicting with P, and $T \overset{s}{\Rightarrow} T'$. Then $P \parallel T \overset{s}{\Rightarrow} P' \parallel T'$. Since P and T are nonconflicting, there exists a completion $t \in \mathbf{A}^*\omega$ such that $P' \parallel T' \overset{t}{\Rightarrow}$. Hence $P' \overset{t}{\Rightarrow}$, i.e., $t \in \mathcal{M}(P')$ by definition of $\mathcal{M}(P')$, and $T' \overset{t}{\Rightarrow}$. □

A process can have more nonconflicting completions. As mentioned above, the nonconflicting completion semantics is *upward closed*, i.e., if $(s, \mathcal{C}) \in \mathrm{CC}(P)$, then it immediately follows that $(s, \mathcal{C}') \in \mathrm{CC}(P)$ for any language $\mathcal{C}' \supseteq \mathcal{C}$. The following example shows that a process can have other nonconflicting completions still.

Example 3 In order to be nonconflicting with process P_3 in figure 2, a test must initially be able to accept at least one of the traces $\alpha\beta, \alpha\alpha\beta, \alpha\alpha\alpha\beta, \ldots$ Therefore, $\mathrm{CC}(P_3)$ contains the pair $(\varepsilon, \{\alpha\alpha^*\beta\omega\})$. Furthermore, any such test must be able to execute α in its initial state, and any test executing α initially must also be able to cope with P_3 being put back to its initial state by executing the selfloop in the initial state. Therefore, such a test also has to accept at least one of the traces $\alpha\alpha\beta, \alpha\alpha\alpha\beta, \alpha\alpha\alpha\alpha\beta, \ldots$ in its initial state. It follows that $\mathrm{CC}(P_3)$ contains all the pairs $(\varepsilon, \{\alpha^n\alpha^*\beta\omega\})$ for $n \geq 1$.

This example shows that nonconflicting completions can be proper subsets of the success languages of subprocesses. In addition, it points out that, given a trace s, there does not necessarily exist a minimal nonconflicting completion. Therefore, not every nonconflicting completion semantics can be characterised by a set of minimal nonconflicting completions as suggested in (14).

The nonconflicting completion semantics is an accurate characterisation of the conflict preorder. The following result shows that, for all processes, being less conflicting is equivalent to having fewer conditions in their nonconflicting completion semantics. It follows that two processes are conflict equivalent if and only if their nonconflicting completion semantics coincide.

Theorem 6 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$. Then $P_1 \lesssim_{\text{conf}} P_2$ if and only if $\text{CC}(P_1) \subseteq \text{CC}(P_2)$.

Proof. First, assume that $P_1 \lesssim_{\text{conf}} P_2$, and let $(s, \mathcal{C}) \in \text{CC}(P_1)$. Then $\mathcal{C} \subseteq \mathbf{A}^*\omega$ is a nonconflicting completion for s in P_1. To prove that $\mathcal{C}$ is a nonconflicting completion for s in P_2, let $T \in \Pi_{\mathbf{A}}$ be nonconflicting with P_2, and $T \overset{s}{\Rightarrow} T'$. Since $P_1 \lesssim_{\text{conf}} P_2$, it follows that P_1 and T are nonconflicting. Since $T \overset{s}{\Rightarrow} T'$ and $\mathcal{C}$ is a nonconflicting completion for s in P_1, there exists a completion $t \in \mathcal{C}$ such that $T' \overset{t}{\Rightarrow}$. Therefore $\mathcal{C}$ is a nonconflicting completion for s in P_2.

Second, assume that $\text{CC}(P_1) \subseteq \text{CC}(P_2)$, and let $T \in \Pi_{\mathbf{A}}$ be nonconflicting with P_2. To prove that P_1 and T are nonconflicting, let $s \in \mathbf{A}^*$ such that $P_1 \parallel T \overset{s}{\Rightarrow} P'$. Then there exist processes $P'_1, T' \in \Pi_{\mathbf{A}}$ such that $P_1 \overset{s}{\Rightarrow} P'_1$, $T \overset{s}{\Rightarrow} T'$, and $P' = P'_1 \parallel T'$. By lemma 5 it follows that

$$(s, \mathcal{M}(P'_1)) \in \text{CC}(P_1) \subseteq \text{CC}(P_2) , \tag{16}$$

i.e., $\mathcal{M}(P'_1)$ is a nonconflicting completion for s in P_2. Then, since P_2 and T are nonconflicting and $T \overset{s}{\Rightarrow} T'$, there exists a completion $t \in \mathcal{M}(P'_1)$ such that $T' \overset{t}{\Rightarrow}$. By definition of $\mathcal{M}(P'_1)$, it follows that $t \in \mathbf{A}^*\omega$ and $P'_1 \overset{t}{\Rightarrow}$. This means $P' = P'_1 \parallel T' \overset{t}{\Rightarrow}$, i.e., P_1 and T are nonconflicting. □

3.5. The Set of Certain Conflicts

Every process can be associated with a language of *certain conflicts*, which characterises its potential to cause conflicts in larger contexts. This language is studied from the viewpoint of discrete event systems in [17]. It can be used to simplify processes in a conflict-preserving way, and to detect conflicts in large systems without constructing their synchronous products explicitly. In the following, the set of certain conflicts is introduced in an alternative way, using the nonconflicting completion semantics.

If $(s, \emptyset) \in \text{CC}(P)$ for some $s \in \mathbf{A}^*$, then any test capable of accepting s and being nonconflicting with P has to be able to continue with a trace from $\emptyset$. Such a test cannot exist. In other words, every test that can accept s is necessarily conflicting with P.

Definition 8 Let $P \in \Pi_{\mathbf{A}}$. Then define

$$\textsc{Conf}(P) \overset{\text{def}}{=} \{ s \in \mathbf{A}^* \mid (s, \emptyset) \in \text{CC}(P) \} ; \tag{17}$$

$$\textsc{NConf}(P) \overset{\text{def}}{=} \{ s \in \mathbf{A}^* \mid (s, \emptyset) \notin \text{CC}(P) \} . \tag{18}$$

The language $\textsc{Conf}(P)$ is called the *set of certain conflicts* of P, because it contains all those traces that necessarily lead to a conflict when accepted by a test running together with P. Its complement $\textsc{NConf}(P)$ contains all those traces that are accepted by some test which is nonconflicting with P.

For nonblocking processes, the set of certain conflicts obviously is empty. Yet, the concept becomes useful and interesting when blocking processes are considered.

Example 4 Consider the processes in figure 3. Any test that initially executes β is conflicting with P_4, and likewise with P_4'. Therefore $\textsc{Conf}(P_4) = \beta\mathbf{A}^*$. The situation is similar for P_5, where any test initially executing α is in conflict. But actually any test that is to be nonconflicting with P_5 must initially accept α to make P_5 reach its only marked state. Therefore $\textsc{Conf}(P_5) = \mathbf{A}^*$.

An immediate consequence of the definition is that the set $\textsc{Conf}(P)$ of certain conflicts decreases when a process becomes less conflicting, and that $\textsc{NConf}(P)$ increases at the same time.

Proposition 7 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$ such that $P_1 \lesssim_{\text{conf}} P_2$. Then

$$\begin{aligned} \textsc{Conf}(P_1) &\subseteq \textsc{Conf}(P_2)\ ; && (19)\\ \textsc{NConf}(P_1) &\supseteq \textsc{NConf}(P_2)\ . && (20) \end{aligned}$$

Proof. Follows immediately from definition 8 and theorem 6. □

The language $\textsc{NConf}(P)$ defines a most general behaviour that can be accepted by any test that is to be nonconflicting with a given process P. Therefore, it can be used to construct a most general test that is nonconflicting with P.

Definition 9 Let $P \in \Pi_{\mathbf{A}}$ be a process. Define the process N_P to be the smallest deterministic process such that

$$\mathcal{M}(N_P) = \textsc{NConf}(P)\omega \quad \text{and} \quad \mathcal{L}(N_P) = \overline{\mathcal{M}(N_P)}\ . \tag{21}$$

By this construction, N_P is the deterministic process which accepts all traces in $\textsc{NConf}(P)$ and can terminate successfully after each of them. This is the largest possible behaviour that can be nonconflicting with P. The following result shows that this process is nonconflicting with P; obviously any process that accepts a trace not in $\textsc{NConf}(P)$ accepts a trace in $\textsc{Conf}(P)$, and therefore is conflicting with P.

Proposition 8 Let $P \in \Pi_{\mathbf{A}}$ be a process. Then P and N_P are nonconflicting.

Proof. Let $s \in \mathbf{A}^*$ such that $P \parallel N_P \overset{s}{\Rightarrow} P' \parallel N_P'$. Then $s \in \mathcal{L}(N_P)$, and by definition of N_P it follows that $s \in \textsc{NConf}(P)$. Thus, there exists a test $T \in \Pi_{\mathbf{A}}$ such that $T \overset{s}{\Rightarrow} T'$ and T is nonconflicting with P. Then $P \parallel T \overset{s}{\Rightarrow} P' \parallel T'$, and there exists $t \in \mathbf{A}^*\omega$ such that $P' \parallel T' \overset{t}{\Rightarrow}$. This means that $st \in \mathcal{M}(P)$ and $st \in \mathcal{L}(T)$. The latter implies $st \in \textsc{NConf}(P)\omega = \mathcal{M}(N_P)$. Since N_P is deterministic and $N_P \overset{s}{\Rightarrow} N_P'$, this implies $N_P' \overset{t}{\Rightarrow}$. Hence $P' \parallel N_P' \overset{t}{\Rightarrow}$, i.e., P and N_P are nonconflicting. □

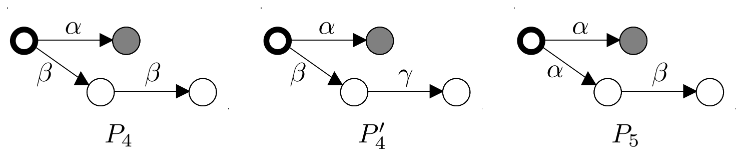

Fig. 3. Examples of blocking processes.

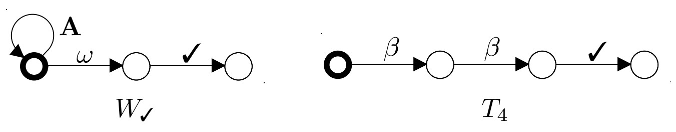

Fig. 4. Example tests used for fair testing.

3.6. Relationship to Fair Testing

Conflict equivalence can be considered as an extension of *fair testing*, which has been introduced in [3, 4, 21] as a testing theory that treats divergence under the assumption of fairness. This section explores the precise relationship between conflicts and fair testing. It is shown that the conflict preorder is coarser than the fair testing preorder.

The conflict and fair testing preorders are closely related and defined in very similar ways—the only difference between them is that the conflict preorder allows both processes and tests to synchronise on the termination action ω. In traditional testing scenarios, like fair testing, success is determined solely by the test, making it impossible to describe blocking processes directly. In fact, blocking processes are exactly the case where the conflict preorder differs from fair testing.

Fair testing [3] uses a relation P **should** T to define when a process P passes a test T. This relation is very similar to P and T being nonconflicting, except that in fair testing only tests may determine whether success occurs or not.

To compare the two preorders and establish the result of this section, the special *fair testing success action* ✓ is introduced. Then the set $\Upsilon_{\mathbf{A}}$ of *fair testing tests* contains processes that make arbitrary use of all actions, including the termination action ω and the success action ✓. Processes are still assumed to use ω as a termination action leading to the special state $\perp$, and they cannot synchronise on the success action ✓.

Definition 10 Let $P \in \Pi_{\mathbf{A}}$ be a process, and $T \in \Upsilon_{\mathbf{A}}$ be a test. Write P **should** T if for every $s \in \mathbf{A}_{\omega}^{*}$ and every P' such that $P \parallel T \overset{s}{\Rightarrow} P'$ there exists $t \in \mathbf{A}_{\omega}^{*}$ such that $P' \overset{t✓}{\Rightarrow}$.

Definition 11 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$ be two processes. The *fair testing preorder* is defined by $P_1 \lesssim_{\text{should}} P_2$ if P_2 **should** T implies P_1 **should** T for every $T \in \Upsilon_{\mathbf{A}}$.

Using this terminology, it can be shown that the conflict preorder is coarser than the fair testing preorder.

Proposition 9 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$. If $P_1 \lesssim_{\text{should}} P_2$ then $P_1 \lesssim_{\text{conf}} P_2$.

Proof. Let $P_1 \lesssim_{\text{should}} P_2$, and let $T \in \Pi_{\mathbf{A}}$ such that P_2 and T are nonconflicting. Consider the test $T_{\checkmark} = T \| W_{\checkmark}$, where $W_{\checkmark}$ is shown in figure 4. $T_{\checkmark}$ succeeds with respect to fair testing whenever T has executed ω. Since P_2 and T are nonconflicting, it follows that P_2 **should** $T_{\checkmark}$. Since $P_1 \lesssim_{\text{should}} P_2$, this implies P_1 **should** $T_{\checkmark}$. Let $s \in \mathbf{A}^*$ such that $P_1 \parallel T \overset{s}{\Rightarrow} P_1' \parallel T'$. This implies $P_1 \parallel T_{\checkmark} \overset{s}{\Rightarrow} P_1' \parallel T_{\checkmark}'$. Since P_1 **should** $T_{\checkmark}$, there exists $t \in \mathbf{A}_\omega^*$ such that $P_1' \parallel T_{\checkmark}' \overset{t\checkmark}{\Rightarrow}$. By construction of $T_{\checkmark}$, it follows that $t = t'\omega$, i.e., $P_1' \parallel T' \overset{t'\omega}{\Rightarrow}$ for some $t' \in \mathbf{A}^*$. Therefore, P_1 and T are nonconflicting, i.e., $P_1 \lesssim_{\text{conf}} P_2$. $\square$

The converse of proposition 9 does not hold. The following example shows that conflict equivalence differs from fair testing equivalence.

Example 5 Processes P_4 and P_4' in figure 3 are conflict equivalent, since exactly those tests that can initially execute β are conflicting with either process. But regarding fair testing, they can be distinguished by test T_4 given in figure 4. Clearly, P_4 **should** T_4 but not P_4' **should** T_4.

This example shows that the fair testing and conflict equivalences treat blocking processes in different ways. To detect a conflict, it is sufficient to know that a process fails, i.e., blocks when executing some trace, but once it is known that a trace leads to blocking, it is immaterial which continuations of that trace can still be executed. Fair testing makes finer distinctions here. It can be shown that the fair testing and conflict preorders coincide for nonblocking processes. Their difference can be characterised precisely in how they handle the set of certain conflicts.

Discrete event systems theory has been extended to consider multiple termination conditions together [9]. Instead of considering a single termination action ω, this new approach introduces a set Ω of termination actions, and processes are required to be nonconflicting with respect to each termination action from Ω.

In the setting of multiple termination conditions, the results of this section are of particular interest. If two processes are fair testing equivalent, it follows by proposition 9 that they are conflict equivalent with respect to all termination conditions. Thus, fair testing provides the best equivalence in cases where all termination conditions must be treated simultaneously, or where the termination condition is not yet known. Conflict equivalence requires the choice of a particular termination condition, but provides a coarser equivalence in this case.

3.7. Relationship to Other Process Semantics

The linear time–branching time spectrum [25] contains a wide range of existing process equivalences. This section explores the relationship of conflict equivalence with known equivalences and shows that it is situated somewhere between bisimulation and failures equivalence.

One of the finest process equivalences is *bisimulation equivalence* [20], which keeps track of the complete branching structure of process behaviours. This is more than enough to distinguish possible conflicts with other processes. It has been established in [3] that bisimulation equivalent processes are also fair testing equivalent. By proposition 9, this result extends to conflict equivalence. Therefore, any algorithms relying on bisimulation can also be used for conflict analysis.

The other end of the spectrum contains the trace and failures preorders and equivalences as the coarsest relationships between processes. The *trace preorder* simply compares the languages of two processes, while the *failures preorder* compares their failures semantics [12, 25].

Definition 12 The *failures semantics* of $P \in \Pi_{\mathbf{A}}$ is

$$\begin{aligned} \mathrm{F}(P) \;=\; \{\, (s,F) \in \mathbf{A}^* \times \mathbb{P}(\mathbf{A}_\omega) \mid P \stackrel{s}{\Rightarrow} P' \text{ and there does not exist} \\ \text{any } \varphi \in F \text{ such that } P' \xrightarrow{\varphi} \,\} \;. \end{aligned} \tag{22}$$

Failures semantics associates with each trace s its immediate failures, i.e., sets of actions which the process may fail to accept after having executed s. This not only differs from nonconflicting completion semantics in that failures are considered instead of successful completions; failures semantics furthermore considers single actions instead of traces.

It is known [3] that failures equivalence differs from fair testing equivalence. Likewise, it differs from conflict equivalence. For example, processes P_2 and P_2' in figure 1 are failures equivalent but not conflict equivalent. Furthermore, the failures preorder is strictly coarser than the fair testing preorder, and the two preorders coincide for finite processes [21]. This does not hold for conflict equivalence, when blocking processes are taken into account.

Example 6 Processes P_4 and P_4' in figure 3 are conflict equivalent as shown in example 5. But their failures semantics are incomparable, because the two processes fail on different actions after executing β. In fact, even the languages of the two processes are incomparable.

Apparently, the traces and failures of conflict equivalent processes may differ completely in parts of the behaviour which are blocking. To understand the precise relationship between conflicts and failures semantics, the set of certain conflicts needs to be taken into account.

Definition 13 For $P \in \Pi_{\mathbf{A}}$, define the *blocking-traces semantics* $\mathcal{L}_{\mathrm{bl}}(P)$ and the *blocking-failures semantics* $\mathrm{F}_{\mathrm{bl}}(P)$ as follows.

$$\mathcal{L}_{\mathrm{bl}}(P) \;=\; \mathcal{L}(P) \cup \textsc{Conf}(P) \;; \tag{23}$$

$$\mathrm{F}_{\mathrm{bl}}(P) \;=\; \mathrm{F}(P \parallel N_P) \cup \textsc{Conf}(P) \times \mathbb{P}(\mathbf{A}_\omega) \;. \tag{24}$$

The blocking-traces semantics simply saturates the language of a process by adding all its certain conflicts. The modification of failures semantics considers all actions that lead into the set of certain conflicts as failures. In addition, it does not make any distinction for traces that are in the set of certain conflicts. An immediate consequence of the definition is

$$\mathcal{L}_{\mathrm{bl}}(P) \;=\; \{\, s \in \mathbf{A}^* \mid (s,F) \in \mathrm{F}_{\mathrm{bl}}(P) \text{ for some } F \subseteq \mathbf{A}_\omega \,\} \;. \tag{25}$$

The blocking-failures and blocking-traces semantics induce two further preorders, which can be shown to be coarser than the conflict preorder.

Definition 14 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$.

- Write $P_1 \lesssim_{\mathcal{L},\mathrm{bl}} P_2$ if $\mathcal{L}_{\mathrm{bl}}(P_1) \subseteq \mathcal{L}_{\mathrm{bl}}(P_2)$.
- Write $P_1 \lesssim_{\mathrm{F},\mathrm{bl}} P_2$ if $\mathrm{F}_{\mathrm{bl}}(P_1) \subseteq \mathrm{F}_{\mathrm{bl}}(P_2)$.

Proposition 10 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$. If $P_1 \lesssim_{\mathrm{conf}} P_2$ then $P_1 \lesssim_{\mathrm{F},\mathrm{bl}} P_2$.

Proof. Let $(s, F) \in \mathrm{F}_{\mathrm{bl}}(P_1)$. If $s \in \textsc{Conf}(P_2)$, it follows immediately that $(s, F) \in \mathrm{F}_{\mathrm{bl}}(P_2)$. Therefore assume $s \in \mathrm{N}\textsc{Conf}(P_2)$. Then $s \in \mathrm{N}\textsc{Conf}(P_1)$ by proposition 7. This implies $(s, F) \in \mathrm{F}(P_1 \,\|\, N_{P_1})$. Therefore let $P_1 \,\|\, N_{P_1} \overset{s}{\Rightarrow} P_1' \,\|\, N_{P_1}'$ such that there does not exist any $\varphi \in F$ with $P_1' \,\|\, N_{P_1}' \xrightarrow{\varphi}$. Now let $\mathcal{C} = \{\, s\sigma t \in \mathbf{A}^{*\omega} \mid \sigma \in \mathbf{A}_\omega \setminus F \,\}$, and construct a deterministic test $T \in \Pi_{\mathbf{A}}$ such that

$$\mathcal{M}(T) \;=\; \mathrm{N}\textsc{Conf}(P_2)\omega \setminus \mathcal{C} \quad \text{and} \quad \mathcal{L}(T) \;=\; \overline{\mathcal{M}(T)}\ . \tag{26}$$

T can be constructed from N_{P_2} by removing all extensions of s via an action not in F from its behaviour. Therefore $N_{P_2} \,\|\, T = T$ and by proposition 7 also $N_{P_1} \,\|\, T = T$. Also, the behaviour of T after accepting any action sequence of which s is not a prefix is identical to that of N_{P_2}, and $T \overset{s}{\Rightarrow} T'$ for some T'

$P_1 \,\|\, N_{P_1}$ and T are conflicting. To see this, let $\sigma \in \mathbf{A}_\omega$ such that $P_1' \,\|\, N_{P_1}' \xrightarrow{\sigma}$. Then $\sigma \notin F$, i.e., $s\sigma \in \mathcal{C}$ and therefore $T' \xrightarrow{\sigma}$ cannot hold. It follows that P_1 and $N_{P_1} \,\|\, T = T$ are conflicting, and P_2 and T are also conflicting since $P_1 \lesssim_{\mathrm{conf}} P_2$. Because of the close relationship between T and N_{P_2} this conflict must be caused by s, i.e.,

$$P_2 \,\|\, T \overset{s}{\Rightarrow} P_2' \,\|\, T'\ , \tag{27}$$

and there does not exist any $t \in \mathbf{A}^*$ such that $P_2' \,\|\, T' \overset{t\omega}{\Rightarrow}$. Let $N_{P_2} \overset{s}{\Rightarrow} N_{P_2}'$. Note that for $\sigma \in \mathbf{A}$, it cannot be the case that both $P_2' \,\|\, N_{P_2}' \xrightarrow{\sigma}$ and $T' \xrightarrow{\sigma}$. Otherwise

$$P_2' \,\|\, N_{P_2}' \,\|\, T' \xrightarrow{\sigma} P_2'' \,\|\, N_{P_2}'' \,\|\, T'' = P_2'' \,\|\, N_{P_2}'' \overset{u\omega}{\Rightarrow}\ , \tag{28}$$

because P_2 and N_{P_2} are nonconflicting. This means that $P_2' \,\|\, N_{P_2}' \xrightarrow{\sigma}$ implies $s\sigma \in \mathcal{C}$, i.e., $\sigma \notin F$. It follows that $(s, F) \in F(P_2 \,\|\, N_{P_2}) \subseteq \mathrm{F}_{\mathrm{bl}}(P_2)$. □

Corollary 11 Let $P_1, P_2 \in \Pi_{\mathbf{A}}$. If $P_1 \lesssim_{\mathrm{conf}} P_2$ then $P_1 \lesssim_{\mathcal{L},\mathrm{bl}} P_2$.

Proof. Using (25), this result follows immediately from proposition 10. □

These results show that the conflict preorder implies the blocking-failures and the blocking-traces preorders. This generalises the known relationships [21] between the fair testing and failures preorder to the case of blocking processes.

4. Conclusions

The notion of *conflicts* [10] from discrete event systems theory has been introduced, and its relationship to process-algebraic testing semantics has been explored. It has been shown that conflicts induce a testing preorder that is closely related to *fair testing* [3,4,21] but coarser, and that most of the known properties of fair testing apply to this *conflict preorder*. Through the introduction of the *set of certain* *conflicts*, known results about fair testing have been extended to the case of blocking processes. Furthermore, the conflict preorder has been shown to be the best possible preorder for modular reasoning about conflicts.

The problem of checking whether a large system of concurrent processes is nonconflicting remains a challenging research problem, because it is very hard to analyse this property in in a modular way [26]. An alternative to brute-force model checking [19], could be to simplify subsystems using *abstractions* [8, 17] that preserve possible conflicts with the rest of the system.

The results of this paper provide the mathematical framework needed to compute conflict-preserving abstractions of state transition systems. In the future, the authors plan to apply these results to design algorithms that can perform such abstractions automatically, and use them to analyse large discrete event systems in a modular way. The goal is to develop efficient techniques for verifying complex systems to be conflicting or nonconflicting.

Acknowledgements

Thanks to Petra Malik for many helpful discussions. Further thanks to the New Zealand Government's Foundation for Research, Science, and Technology for funding to make this research possible.

References

1. A. Arnold. *Finite Transitions Systems: Semantics of Communicating Systems.* Prentice-Hall, 1994.
2. B. Brandin and F. Charbonnier. The supervisory control of the automated manufacturing system of the AIP. In *Proc. Rensselaer's 4th Int. Conf. Computer Integrated Manufacturing and Automation Technology*, pages 319–324, Troy, NY, USA, 1994.
3. E. Brinksma, A. Rensink, and W. Vogler. Fair testing. In I. Lee and S. A. Smolka, editors, *Proc. 6th Int. Conf. Concurrency Theory, CONCUR '95*, volume 962 of *LNCS*, pages 313–327, Philadelphia, PA, USA, 1995. Springer.
4. E. Brinksma, A. Rensink, and W. Vogler. Applications of fair testing. In R. Gotzhein and J. Bredereke, editors, *Formal Description Techniques IX: Theory, application and tools, IFIP TC6 WG6.1 Int. Conf. Formal Description Techniques IX / Protocol Specification, Testing and Verification XVI*, volume 69 of *IFIP Conf. Proc.*, pages 145–160, Kaiserslautern, Germany, Oct. 1996. Kluwer.
5. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.
6. C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer, Sept. 1999.
7. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Programming Languages and Systems*, 8(2):244–263, Apr. 1986.
8. E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 1999.
9. M. H. de Queiroz, J. E. R. Cury, and W. M. Wonham. Multi-tasking supervisory control of discrete-event systems. In *Proc. 7th Int. Workshop on Discrete Event*

Systems, WODES '04, pages 175–180, Reims, France, Sept. 2004.

10. P. Dietrich, R. Malik, W. M. Wonham, and B. A. Brandin. Implementation considerations in supervisory control. In B. Caillaud, P. Darondeau, L. Lavagno, and X. Xie, editors, *Synthesis and Control of Discrete Event Systems*, pages 185–201. Kluwer, 2002.
11. M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.
12. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
13. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2001.
14. R. J. Leduc, B. A. Brandin, and W. M. Wonham. Hierarchical interface-based non-blocking verification. In *Proc. Canadian Conf. Electrical and Computer Engineering*, pages 1–6, May 2000.
15. R. J. Leduc and W. M. Wonham. PLC implementation of a DES supervisor for a manufacturing testbed. In *Proc. 33rd Allerton Conf. Communication, Control and Computing*, pages 519–528, Monticello, Illinois, Oct. 1995.
16. P. Malik. *From Supervisory Control to Nonblocking Controllers for Discrete Event Systems.* PhD thesis, University of Kaiserslautern, Kaiserslautern, Germany, 2003.
17. R. Malik. On the set of certain conflicts of a given language. In *Proc. 7th Int. Workshop on Discrete Event Systems, WODES '04*, pages 277–282, Reims, France, Sept. 2004.
18. R. Malik, D. Streader, and S. Reeves. Fair testing revisited: A process-algebraic characterisation of conflicts. In F. Wang, editor, *Proc. 2nd Int. Symp. Automated Technology for Verification and Analysis, ATVA 2004*, volume 3299 of *LNCS*, pages 120–134, Taipei, Taiwan, Oct.–Nov. 2004. Springer.
19. K. L. McMillan. *Symbolic Model Checking.* Kluwer, 1993.
20. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS.* Springer, 1980.
21. V. Natarajan and R. Cleaveland. Divergence and fair testing. In *Proc. 22nd Int. Colloquium on Automata, Languages, and Programming, ICALP '95*, pages 648–659, 1995.
22. R. D. Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Comput. Sci.*, 34(1–2):83–133, Nov. 1984.
23. P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. IEEE*, 77(1):81–98, Jan. 1989.
24. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall, 1997.
25. R. J. van Glabbeek. The linear time — branching time spectrum I: The semantics of concrete, sequential processes. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.
26. K. C. Wong, J. G. Thistle, R. P. Malhame, and H.-H. Hoang. Supervisory control of distributed systems: Conflict resolution. *Discrete Event Dynamic Systems: Theory and Applications*, 10:131–186, 2000.
27. K. C. Wong and W. M. Wonham. Modular control and coordination of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 8(3):247–297, Oct. 1998.
28. Z. H. Zhang and W. M. Wonham. STCT: An efficient algorithm for supervisory control design. In B. Caillaud, P. Darondeau, L. Lavagno, and X. Xie, editors, *Synthesis and Control of Discrete Event Systems*, pages 77–100. Kluwer, 2002.