Conformance and Mirroring for Timed Asynchronous Circuits

Bin Zhou
Tokyo Institute of Technology
e-mail: zhou@cs.titech.ac.jp

Tomohiro Yoneda
Tokyo Institute of Technology
yoneda@cs.titech.ac.jp

Bernd-Holger Schlingloff
Universität Bremen, TZi-BISS
hs@tzi.org

Abstract – Conformance has been used as a correctness criterion for asynchronous circuits. In the case of untimed systems, conformance of an implementation to a specification is equivalent to the failure-freeness between the implementation and the mirror of the specification. For bounded-delay systems, in general this property does not hold. In this paper, we define various notions of failure and examine whether the above propery holds or not. We then discuss alternative effective algorithms for conformance checking of bounded-delay asynchronous circuits.

I. Introduction

Conformance of an implementation with respect to a specification was defined as a correctness criterion for reactive systems in Dill [1]. In contrast to other verification methods like temporal logic model checking [2], in the conformance checking approach both specification and implementation are given graphically, e.g., as signal transition graphs or one-safe Petri nets. Since such formalisms are familiar in engineering and computer science, the method is easily accepted in practice. Various verification tools based on conformance checking have been developed [5, 7, 6, 8]. One of the biggest advantages of the approach is that the verification can be done hierarchically. This is essential for the verification of large systems.

A main drawback of the above methods is that it disregards the internal timing of systems. For example, in order to construct fast and compact asynchronous circuits, designers use some *bounded delay* model. That is, for each gate they assume a minimal and maximal bound on its switching time. Circuits designed under such delay assumptions are called *timed circuits*. Since their functionality depends on the real-time constraints, verification techniques which do not consider these constraints can give incorrect results.

Therefore, it is important to develop verification algorithms for such timed systems. Several research projects (e.g., [11, 14]) have succeeded in verifying rather large bounded delay asynchronous circuits. Although all these conformance checking algorithms are based on similar ideas, their notions of correctness differ.

Intuitively, an implementation conforms to a specification if it can safely replace the specification in any possible environment(see Fig.1). That is, the implementation may not fail unless the specification allows a failure in the same context. In order to check this property directly, we would have to consider the countless possible environments, which is impossible. However, for untimed systems, the specification itself forms a "most general environment": the implementation conforms to the specification if the composition of implementation and the mirror of the specification is failure free. We call this property *mirror property*. Clearly, the notion of conformance and the mirror property depend on what we regard as a failure. For untimed systems, *safety failure* [1] is well-understood. In timed systems, additional failures may arise by wrong timing. However, it is much less clear what an intuitive and generally acceptable definition of timing failure could be. In this paper, we define and discuss several alternatives.

In conformance checking for real-time systems, the mirror module defined in [1] is unachivable. If we choose to use the same way (by swap the *input* and the *output*) as used in untimed systems, one of the main problems is that the mirror property does not hold in general. Depending on the chosen definition of failure, it may or may not be possible to implement a conformance checking procedure by mirroring. We give several possible definitions of safety and timing failures, and in each case show why the mirror property fails, or give sufficient conditions under which it holds.

II. Definition of correctness

Let $\mathcal{A}$ be a finite alphabet of *symbols*. Typically, a symbol could be the name of a wire, signal, or action. Subsequently, we assume that each symbol in $\mathcal{A}$ is either an *input symbol*($\mathcal{I}$) or an *output symbol* ($\mathcal{O}$); that is, $\mathcal{A} = \mathcal{I} \cup \mathcal{O}$, and $\mathcal{I} \cap \mathcal{O} = \emptyset$. Let $\mathbf{Q}^+$ be the domain of rational numbers for *time-points*. For any $w \in \mathcal{A}$ and $\tau \in \mathbf{Q}^+$, the pair (w, τ) is called a *(timed) event*. Intuitively, (w, τ) indicates that at time τ, the voltage on wire w changes, signal w arrives, or action w is performed. We say that an output is *sent* or *produced*, and the corresponding input is *received* or *accepted*. A *(timed) trace* x in $\mathcal{A}$ is a finite or infinite sequence of events $x = e_0 e_1 \cdots$, where $e_i = (w_i, \tau_i)$, such that the following properties are satisfied:

- *Monotonicity:* $\tau_i \leq \tau_{i+1}$ for all $i \geq 0$, and
- *Progress:* if x is infinite, then for any $\tau \in \mathbf{Q}^+$ there exists an i such that $\tau_i > \tau$.

The *length* of a trace is the number of events it contains; ε denotes the empty trace of length 0. A *module* is a tuple $M = (\mathcal{I}, \mathcal{O}, \mathcal{T})$, where $\mathcal{A} = \mathcal{I} \cup \mathcal{O}$ is the alphabet, and $\mathcal{T}$ is a set of traces in $\mathcal{A}$. We assume that $\mathcal{T}$ is generated by a time Petri net N, where the transitions of N are labelled by symbols from $\mathcal{A}$. Moreover, we assume that N is deterministic, i.e., no two transitions labelled with the same symbol are enabled at the same time in each reachable state. This restriction is common for conformance checking algorithms.

A set of modules is *composable*, if no two modules in the set have any output symbol in common. Whenever we consider sets of modules in this paper, we assume that they are composable. If $\mathcal{M}_C = \{M_1, \cdots, M_n\}$ is a (composable) set of modules, then we define $\mathcal{O}_C = \bigcup_{k=1}^{n} \mathcal{O}_k$ and $\mathcal{I}_C = \bigcup_{k=1}^{n} \mathcal{I}_k - \mathcal{O}_C$.

Symbols in $\bigcup_{k=1}^{n} \mathcal{O}_k \cap \bigcup_{k=1}^{n} \mathcal{I}_k$ are called *internal*; they can also be observed as an output of the composition.

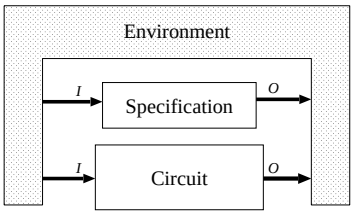

Fig. 1. Correctness of a circuit with respect to a specification.

We say that module $M_E = (\mathcal{I}_E, \mathcal{O}_E, \mathcal{T}_E)$ is an *admissible environment* for $\mathcal{M}_C$, if M_C and M_E are composable, $\mathcal{O}_E \supseteq \mathcal{I}_C$ and $\mathcal{I}_E \subseteq \mathcal{O}_C$. Both the environment and the implementation module may have additional outputs which are not observed by the other one. We say that $M_S = (\mathcal{I}_S, \mathcal{O}_S, \mathcal{T}_S)$ is an *admissible specification* for $\mathcal{M}_C$, if $\mathcal{I}_S \supseteq \mathcal{I}_C$, $\mathcal{O}_S \subseteq \mathcal{O}_C$, and $\mathcal{I}_S \cap \mathcal{O}_C = \emptyset$. That is, an admissible specification describes a causal relation between all of the implementation's inputs and some of its outputs. The specification can provide more input stimuli than the implementation actually uses, and the implementation can have several extra outputs whose behaviors are not constrained by the specification.

Formally, a *failure* between modules is just a trace in the modules' alphabet. Later on, we will define several types of failures between modules. Dependent on the respective definition of failure, we define *conformance* as a notion of correctness of an implementation with respect to a specification.

Definition 1 Suppose that a system is modeled by a set $\mathcal{M}_C = \{M_1, \cdots, M_n\}$ of modules, and that an admissible specification for this system is given by module $M_S = (\mathcal{I}_S, \mathcal{O}_S, \mathcal{T}_S)$. We say that $\mathcal{M}_C$ *conforms to* M_S, if in any admissible environment M_E for both $\mathcal{M}_C$ and M_S such that $\{M_S, M_E\}$ is failure-free, $\mathcal{M}_C \cup \{M_E\}$ is also failure-free.

If $\mathcal{M}_C$ conforms to M_S, then $\mathcal{M}_C$ may fail in an environment only if the specification allows a failure in this environment (See Fig.1). Checking this property is our verification task.

Correctness defined by conformance is different from trace inclusion, (bi-)simulation or logical implication. A common class of failures is that after a sequence of actions, one module in a system produces an event but other modules can not accept it. Consider modules M_{S1} and M_C shown in Fig. 2(a). M_C does not conform to M_{S1}, because there exists a module M_{E1} shown in Fig. 2(b) which has no failure with M_{S1} but has a failure $((b,1)(x,2))$ with M_C. On the other hand, consider M_{S2} in Fig. 2 (c). In this case, M_{S2} is not interested in receiving an input b. Therefore, M_{E1} now has a failure with M_{S2}. A module which has no failure with M_{S2} is, for example, M_{E2} in Fig. 2(d). It does not have any failure with M_C either. Actually, any M_E which has no failure with M_{S2}, also has no failure with M_C as long as it is an admissible environment for M_{S2} and M_C. Hence, M_C conforms to M_{S2}. Note that M_C implements the additional behavior "bx" which M_{S2} does not care about. We consider M_C to be correct with respect to M_{S2}, because M_C correctly implements the behavior specified by M_{S2}, even if M_C is not a bisimulation of M_{S2}, and the trace set of M_C is not included in that of M_{S2}. This is an important property in circuit verification, because circuits usually implement some additional functionalities for inputs not constrained by its specification.

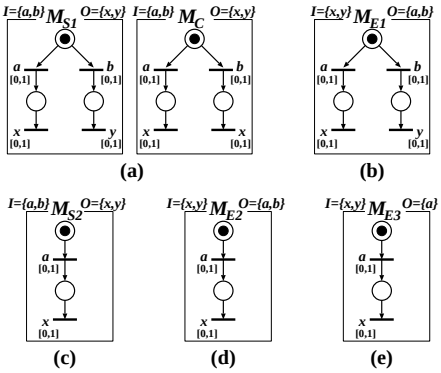

Fig. 2. Examples of failures.

In the definition of conformance, only admissible environments are considered for the following reason. Consider the environment M_{E3} shown in Fig. 2(e). Note that M_{E3} does not have b as an output symbol, and, hence, is not admissible. Here, M_{E3} has the failure $((b,0)(x,1))$ with M_C : both M_C and M_{E3} admit the one-element trace $((b,0))$, because M_C accepts it, and M_{E3} ignores b. Thus, after the trace $((b,0))$, module M_C may decide to produce an output x at time 1, but the environment M_{E3} can not accept it (note that M_{E3} must produce an a before being ready to receive x). On the other hand, $\{M_{S2}, M_{E3}\}$ is failure-free (M_{S2} does not accept the trace $((b,0))$). We conclude that without admissibility constraints on the environment, M_C would not conform to M_{S2}. In this sense, omitting these constraints in the definition of conformance would restrict the possibility of additional functionality for a correct implementation.

A major benefit of using conformance as a correctness criterion is that the verification can be done hierarchically. This allows us to verify large systems, which are built from component libraries in a hierarchical way.

Theorem 1 If $\{M_1, \cdots, M_{k-1}, M_k, M_{k+1}, \cdots, M_n\}$ conforms to M_S, $\{M_{k_1}, \cdots, M_{k_m}\}$ conforms to M_k, and $(\mathcal{A}_S \cup \bigcup_{i=1}^{n} \mathcal{A}_i - \mathcal{A}_k) \cap \bigcup_{i=1}^{m} \mathcal{A}_{k_i} = \emptyset$ holds, then $\{M_1, \cdots, M_{k-1}, M_{k_1}, \cdots, M_{k_m}, M_{k+1}, \cdots, M_n\}$ conforms to M_S, where $M_S = (\mathcal{I}_S, \mathcal{O}_S, \mathcal{T}_S)$, $M_i = (\mathcal{I}_i, \mathcal{O}_i, \mathcal{T}_i)$, $M_{k_i} = (\mathcal{I}_{k_i}, \mathcal{O}_{k_i}, \mathcal{T}_{k_i})$, $\mathcal{A}_S = \mathcal{I}_S \cup \mathcal{O}_S$, $\mathcal{A}_i = \mathcal{I}_i \cup \mathcal{O}_i$, and $\mathcal{A}_{k_i} = \mathcal{I}_{k_i} \cup \mathcal{O}_{k_i}$.

The proof is omitted because of limitations of space. All the proofs of the theorems shown in this paper can be found in the full version of our paper[15].

To implement a conformance checking algorithm, Definition 1 is not well suited: in general it is not possible to construct every admissible M_E which has no failure with M_S. Let the *mirror module* M^m of a module $M = (\mathcal{I}, \mathcal{O}, \mathcal{T})$ be defined as $M^m = (\mathcal{O}, \mathcal{I}, \mathcal{T})$. For untimed system verification, in [1] it is shown that the following property holds.

[Mirror property] $\mathcal{M}_C$ conforms to its admissible specification M_S, iff $\mathcal{M}_C \cup \{M_S^m\}$ is failure-free.

If the mirror property holds, then it is easy to implement a conformance checking procedure without considering all possible environments M_E: we just need to construct the set of

failures of $\mathcal{M}_C \cup \{M_S^m\}$. Unfortunately, for timed system verification, the mirror property does not hold in general. In the following two sections, we will discuss conditions that the mirror property holds.

III. Checking safety properties

In this section, we consider safety properties. We use the definition of safety failures from [12, 13, 14]. In the next section, we extend the definition such that timing properties can be checked.

The *projection* of a trace x in $\mathcal{A}$ onto another alphabet $\mathcal{A}'$ can be defined as usual; assume that $x = e_1e_2e_3\cdots$ and $e_1 = (w_1, \tau_1)$,

$$\mathsf{project}(x, \mathcal{A}') = \begin{cases} e_1 y, & \text{if } w_1 \in \mathcal{A}' \\ y & \text{else,} \end{cases}$$

where $y = \mathsf{project}(e_2e_3\cdots, \mathcal{A}')$.

For a trace x and modules $M(= (\mathcal{I}, \mathcal{O}, \mathcal{T}))$, M_1, and M_2, we say that M *admits* x ($M \models x$), if $\mathsf{project}(x, \mathcal{I}\cup\mathcal{O}) \in \mathcal{T}$, and $M_1 \cap M_2 \models x$, if $M_1 \models x$ and $M_2 \models x$. For example, in Fig. 3 it holds that $M_S^m \cap M_I \models ((w,5)(v,11))$ and $M_I \models ((a,1)(u,9))$. Note that in the later example, $(a,1)$ is projected out since $a \notin \mathcal{I}_I \cup \mathcal{O}_I$. A module M *can wait* after trace x *till time* τ, denoted by $M \mid\!\!\vdash x \circ \tau$, if $M \models x(w,\tau')$ for some $\tau' < \tau$ implies that $M \models x(w,\tau'')$ for some $\tau'' \geq \tau$. This means that after admitting x, module M can be inactive till time τ without sending or receiving any events. $\{M_1, ..., M_n\} \mid\!\!\vdash x \circ \tau$ means that $M_i \mid\!\!\vdash x \circ \tau$ for all $i \leq n$. In the example, $M_I \mid\!\!\vdash \varepsilon \circ 8$, but $\{M_S^m, M_I\} \not\mid\!\!\vdash \varepsilon \circ 8$, because M_S^m must produce w before time 5 and thus can not be inactive till time 8. Furthermore, we say that M is *unbounded*, if for any trace x such that $M \models x$, module M can wait after x for an arbitrary amount of time.

Definition 2 Given a set $\mathcal{M} = \{M_1, \cdots, M_n\}$ of modules, where $M_k = (\mathcal{I}_k, \mathcal{O}_k, \mathcal{T}_k)$, a *safety failure* of $\mathcal{M}$ is a finite trace $x = y(w,\tau)$, where $w \in \mathcal{O}_k$ for some $k \leq n$, such that $\mathcal{M} \models y$, $\mathcal{M} \mid\!\!\vdash y \circ \tau$ and $M_k \models x$, but $\mathcal{M} \not\models x$ holds.

We denote the class of all safety failures of $\mathcal{M}$ by $\mathsf{failure0}(\mathcal{M})$. A set $\mathcal{M}$ of modules is *safety failure-free* if $\mathsf{failure0}(\mathcal{M}) = \emptyset$. If $\mathcal{M} = \{M_0, M_1, M_2\}$ and $\mathcal{M}' = \{M_1, M_2\}$, we write $\mathsf{failure0}(M_0, M_1, M_2)$ or $\mathsf{failure0}(M_0, \mathcal{M}')$ for $\mathsf{failure0}(\mathcal{M})$ etc.

Intuitively, a safety failure occurs if after trace x one module M_k sends an output w at time τ, but some other module cannot receive the corresponding input. We do not regard $y(w,\tau)$ as a safety failure of $\mathcal{M}$ if y is not a trace of $\mathcal{M}$, or if after trace y some module M_j of $\mathcal{M}$ must admit some symbol w' before τ (i.e., if $\mathcal{M}$ can not wait till time τ after y): in this case, the event (w',τ') will change the state of M_j and $\mathcal{M}$ may be able to accept (w,τ) after accepting $y(w',\tau')$.

In the example in Fig. 3, the one-element trace $((u,8))$ is a safety failure of $\{M_E, M_I\}$, but there are no safety failures in $\{M_S^m, M_I\}$. From this example it follows that for the class of safety failures, the mirror property does not hold. Since in $\{M_E, M_S\}$ no outputs are enabled and safety failures only occur when an output is sent, $\mathsf{failure0}(M_E, M_S) = \emptyset$. $((u,8)) \in \mathsf{failure0}(M_E, M_I)$. From the definition of conformance, it follows that M_I does not conform to M_S. However, $\mathsf{failure0}(M_S^m, M_I) = \emptyset$. Hence, the mirror property fails for this example.

A sufficient condition that the mirror property holds with respect to safety failures is that specifications are unbounded. That is, we have the following theorem.

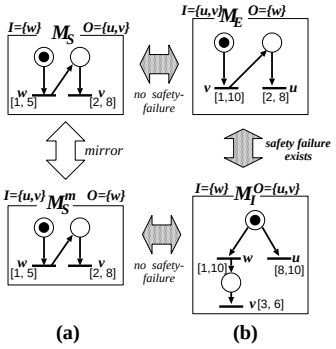

Fig. 3. Mirror property defined by failure0 does not hold.

Theorem 2 If the specification is unbounded, then the mirror property defined by failure0 holds.

This condition restricts the range of safety properties which can be verified by conformance checking. However, in many cases we only want to check the *causal dependency* between events of timed systems. Usually, this can be expressed by an unbounded specification.

IV. Checking both safety and timing

If a system conforms to a specification with respect to safety failures, the system accepts or ignores any input that the specification accepts. However, the system may produce no output even if the specification requires it. That is, even if it is safety failure-free, a system might not satisfy desired timing properties ("it is always safe to do nothing"). Reconsider the example of Fig. 3: in the system $\{M_S, M_E\}$, initially M_E waits for input v from M_S, and M_S waits for input w from M_E. Clearly, this deadlock situation should be considered as a failure. Thus, we need a notion of failure which occurs when some module expects an input, and the corresponding output is not produced in time. In this section, in addition to safety failure, we define six different types of timing failures, which might be useful in practice. Here, we make no claim as to which of these definitions is most intuitive or adequate; our intention is to establish for each of these possible definitions whether the mirror property holds or not. We believe that the ultimate decision on what should be regarded as a timing failure depends on the specific application domain and modelling formalism.

First, for module M, trace y such that $M \models y$, and symbol w, define $latest(y, w, M)$ such that $latest(y, w, M) = \tau + Lft(t)$, if a transition t labelled by w is enabled in M after y and t got enabled most recently at time τ in y. Here, $Lft(t)$ represents the *latest firing time* of transition t, for example, $Lft(u) = 10$ in M_I of Fig. 3. Intuitively, $latest(y, w, M)$ represents the latest time when w can occur in M after y, if it is not in conflict with other transitions. If no t labelled w is enabled in M after y, we define that $latest(y, w, M) = \infty$.

Definition 3 Given a set $\mathcal{M} = \{M_1, \cdots, M_n\}$ of modules, where $M_k = (\mathcal{I}_k, \mathcal{O}_k, \mathcal{T}_k)$, a *type* $\boldsymbol{n}$ *timing failure situation* of $\mathcal{M}$ is a finite trace $x = y(w,\tau)$, where $w \in \mathcal{I}_k$ for some $k \leq n$, such that $\mathcal{M} \models y$, $M_k \models x$, and Condition $\boldsymbol{n}$ is satisfied.

Condition 1: $\mathcal{M} \mid\!\!\vdash y \circ \tau$, but $\mathcal{M} \not\models x$ holds.

Condition 2: $\mathcal{M} \mid\!\!\vdash y \circ \tau'$ for every τ' such that $M_k \models y(w, \tau')$, but there does not exist τ'' such that $\mathcal{M} \models y(w, \tau'')$ holds.

Condition 3: There are no j, w' and τ' such that $w' \in \mathcal{O}_j$ and $\mathcal{M} \models y(w', \tau')$ hold.

Condition 4: There are no j, w' and τ' such that $j \neq k$, $w' \in \mathcal{O}_j$ and $\mathcal{M} \models y(w', \tau')$ hold.

Condition 5: There are no j, w' and τ' such that $w' \in \mathcal{O}_j$, $\mathcal{M} \models y(w', \tau')$ and $latest(y, w', M_j) \leq latest(y, w, M_k)$ hold.

Condition 6: There are no j, w' and τ' such that $j \neq k$, $w' \in \mathcal{O}_j$, $\mathcal{M} \models y(w', \tau')$ and $latest(y, w', M_j) \leq latest(y, w, M_k)$ hold.

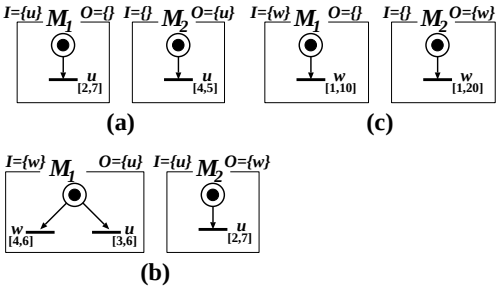

Fig. 4. Examples of module sets.

Intuitively, a timing failure situation occurs when a module expects an input, but it will not be given in time. Here, there are several interpretations on "in time". For simply, we use *timing failure* instead of *timing failure situation* in this paper. In type 1 timing failures, it is the exact time (τ) when the input is expected. That is, for each time point at which the receiver can accept a symbol the sender must be able to produce it. Thus, for example, $(u, 3)$ is a type 1 timing failure of M_1 and M_2 shown in Fig. 4(a), while $(u, 4)$ is not. Note that failure-freeness of $\{M_1, M_2\}$ with respect to both safety and type 1 timing failure is "almost" the same as trace equivalence of M_1 and M_2^m.

In type 2 timing failures, it is the whole time period when the receiver can accept the input. That is, if the receiver expects an input within a certain time interval, the sender must be able to send it at least at one time point within this interval. If w can be produced at any one time point where M_k can accept it, then no type 2 timing failure occurs. Thus, the above M_1 and M_2 have no type 2 timing failures (M_1 expects input u between time 2 and 7, and M_2 can send it, e.g., at time 5). As in the case of safety failure, if a module must change its state before the requested time, then there can not be a failure in this state. For example, $(u, 6)$ is not a type 1 timing failure of M_1 and M_2, because M_2 must make a transition at latest at time 5 and can not wait till time 6.

In Conditions 3 to 6, to avoid a timing failure, it is not necessary to send exactly the symbol w that the receiver expects. Instead, *some* output (w') must be produced and accepted in time. This reflects the consideration that in order to avoid a deadlock, it suffices to always be able to send at least one of the inputs the other partner is waiting for. However, if an alternative output (w') is sent and the former input (w) is still expected, the condition must hold again; therefore, timing failure freeness according to these conditions implies that if an input is continually expected it must finally be given.

In type 3 and 5, we allow that the alternative output is produced by *any* module (including the one which expects the input), whereas in type 4 and 6 the alternative symbol must be produced by some *other* module. The difference between Condition 2, 4 and 6, and Condition 3 and 5 is illustrated in Fig. 4(b), where the input w expected by M_1 can be disabled by sending a conflicting output u. In this example, $(w, 6)$ is a type 2 timing failure (M_1 expects input w between time 4 and 6, but M_2 does not send it). Also, there is no alternative output sent by M_2, therefore it is also a timing failure according to Condition 4 and 6. On the other hand, $\{M_1, M_2\}$ is timing failure free according to Condition 3 or 5, since, here we assume that if it doesn't get the expected input w in time, M_1 can choose to produce the output u instead.

In Condition 5 and 6, we require an alternative output which *must* be sent before the latest time at which the input is expected. Intuitively, if a module expects an input within a certain time which is not provided, this is regarded as a failure even if an alternative output can occur at some later time. The difference between timing failures of type 3 and 4 and timing failures of type 5 and 6 is illustrated in Fig. 4(c). In this example, $((w, 5))$ is a type 5 and 6 timing failure (M_1 expects w before time 10, but M_2 may choose to send it only at time 20). However, $\{M_1, M_2\}$ has no timing failures of type 1 to 4, since for each time point $\tau \leq 10$ at which M_1 expects w, M_2 can produce w, and it is safety failure free, since $\{M_1, M_2\}$ can only wait till time 10.

There are still other possible definitions of timing failures which we considered; however, the above list seems to contain most of our intuitive ideas about the notion of timing failure in different application contexts. Let $\mathsf{failure}n(M_1, \cdots, M_n)$ be the union of all safety failures and type n timing failures of $\{M_1, \cdots, M_n\}$. Next, we will exhibit counterexamples to the mirror property for various failures.

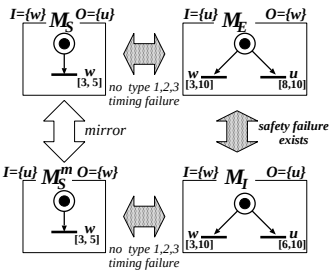

Fig. 5. A counterexample to the mirror property for failure $1, 2, 3$.

Consider the example shown in Fig. 5. Neither $\{M_S, M_E\}$ nor $\{M_S^m, M_I\}$ has any safety failures, because w is accepted by the receiver, and M_S or M_S^m can only wait till time 5. $\{M_I, M_E\}$ has a safety failure $(u, 6)$, because $M_I \models (u, 6)$, $M_E \mid\!\!\vdash \varepsilon \circ 6$, but $(u, 6) \notin M_E$. Concerning timing failures, $\mathsf{failure}n(M_S, M_E) = \emptyset$ holds for $n \in \{1, 2, 3\}$, because the input w of M_S is given by M_E in time, and for the input u of M_E, for example, $(u, 8)$, initially M_S can not wait till time 8. $(u, 8)$ is both a type 4 and 6 timing failure, because M_E itself produces the other output w. $(w, 4)$ is both a type 5 and 6 timing failure from $latest(\varepsilon, w, M_E) > latest(\varepsilon, w, M_S)$. For M_S^m and M_I, $\mathsf{failure}n(M_S^m, M_I) = \emptyset$ holds for any n, because w of M_I

is the only input and it is given by M_S^m in time. Although $\{M_I, M_E\}$ is timing failure-free, failure$n(M_I, M_E) \neq \emptyset$ for any n due to the safety failures of them (remember that failure$n$ includes safety failures). Hence, the mirror properties defined by failure1, 2, 3 do not hold in this example, because for $n \in \{1, 2, 3\}$, failure$n(M_S, M_E) = \emptyset$ and failure$n(M_I, M_E) \neq \emptyset$ (i.e., M_I does not conform to M_S), but failure$n(M_S^m, M_I) = \emptyset$.

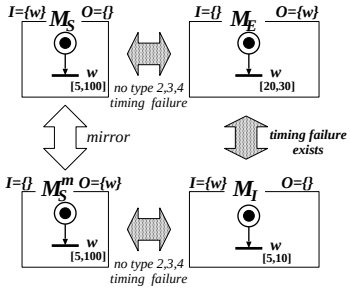

Fig. 6. A counterexample to the mirror property for failure $2, 3, 4$.

In the example shown in Fig. 6, no safety failures exist in each pair of modules. For $n \in \{2, 3, 4\}$, failure$n(M_S, M_E) =$ failure$n(M_S^m, M_I) = \emptyset$, $(w, 10)$ is in failure1(M_S, M_E), failure6(M_S^m, M_I) and failure5(M_S^m, M_I). The latter two are due to $latest(\varepsilon, w, M_S^m) > latest(\varepsilon, w, M_I)$. On the other hand, $(w, 10) \in$ failure$n(M_E, M_I)$ for $n \in \{1, 2, 3, 4, 5, 6\}$. Therefore, M_I does not conform to M_S with respect to failure$2, 3, 4$. Hence, the mirror properties defined by failure$2, 3, 4$ do not hold in this example.

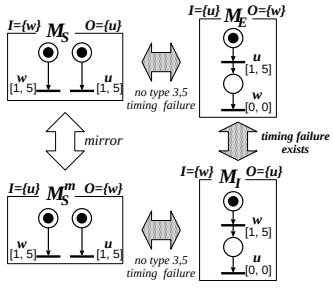

Fig. 7. A counterexample to the mirror property for failure $3, 5$.

Fig. 7 shows an example in which the mirror properties defined by failure$3, 5$ do not hold. Similar to the previous example, here each pair of modules is safety failure free. Since no output symbols can change in M_E and M_I, $\{M_E, M_I\}$ has every type of timing failures. On the other hand, failure5$(M_S, M_E) = \emptyset$ holds, because for the expected input w of M_S, some other output u is produced in time (i.e., $latest(\varepsilon, u, M_S) \leq latest(\varepsilon, w, M_S)$), and after receiving u, M_E immediately produce w, which is also in time again from $latest((u, \tau_u), w, M_E) \leq latest((u, \tau_u), w, M_S)$. This also implies failure3$(M_S, M_E) = \emptyset$. Similarly, we can derive failure$n(M_S^m, M_I) = \emptyset$ for $n \in \{3, 5\}$. Hence, the mirror properties defined by failure$3, 5$ do not hold. failure$n(M_S, M_E) \neq \emptyset$ for $n \in \{1, 2\}$, because M_E can not produce w without receiving u, and, failure$n(M_S, M_E) \neq \emptyset$ for $n \in \{4, 6\}$, because the same module M_S produces the other output u.

So far, we have given counterexamples to the mirror properties defined by failure1 to 5. Surprisingly, after several failed attempts to construct a counterexample to the mirror property defined by failure6, we could prove the following theorem.

Theorem 3 The mirror property defined by failure6 holds.

This theorem shows that $latest(y, w', M_j) \leq latest(y, w, M_k)$ and $j \neq k$ are important conditions in the definition of timing failures.

Next, we will consider additional conditions such that the mirror property holds for the various notions of failures. Since type 1 timing failures are very similar to safety failures, Theorem 2 indicates that unboundedness of the specification plays an important role.

Theorem 4 If the specification is unbounded, then the mirror property defined by failure1 holds.

Unboundedness of specifications is not sufficient for the mirror properties of the other failures. Actually, in the examples in Fig. 6 and Fig. 7, even if the latest firing times of u and w in M_S are increased up to infinity, still failure$n(M_S, M_E) =$ failure$n(M_S^m, M_I) = \emptyset$ and failure$n(M_E, M_I) \neq \emptyset$ hold for $n \in \{2, 3, 4, 5\}$.

If the inputs and outputs in a specification cannot be fired simultaneously, i.e., for any y such that $M_S \models y$, there do not exist $u \in \mathcal{I}_S$ and $w \in \mathcal{O}_S$ such that both $M_S \models y(u, \tau)$ and $M_S \models y(w, \tau)$ hold, then we say that the specification is I/O *conflict free*. I/O conflict freeness of a specification implies that some other module has to produce an output for the input expected by the specification. Thus, this is very similar to the Condition 6 in the definition of timing failures. However, as shown in the next theorem, only specifications need this condition for the mirror property.

Theorem 5 If the specification is I/O conflict free, then the mirror property defined by failure5 holds.

Since the specifications in Fig. 5 and 6 are I/O conflict free, this restriction is effective only for the mirror property defined by failure5. In many application domains (e.g., for the purpose of formal testing), specifications can always be written such that they are I/O conflict free. Therefore, in such domains conformance checking(w.r.t. Condition 5) can be implemented by mirroring.

For the verification of timed asynchronous circuits, it can be argued that type 5 timing failures are most appropriate. However, in this context we can not always guarantee that specifications are I/O conflict free. Thus, in [13] we started to develop techniques for conformance checking (w.r.t. failure5). Here, we further elaborate these ideas and explain them with an appropriate example. We also give an improved version of the main theorem and proof of [13] concerning the correctness of the algorithms in [15].

For a set $\mathcal{M} = \{M_1, \cdots, M_n\}$ of modules and a trace x, we call $w \in \mathcal{O}_j$ a *limited output*, if $latest(x, w, M_j) \leq latest(x, w', M_k)$ holds for any k and $w' \in \mathcal{O}_k$. That is, w decides the latest time point up to which $\mathcal{M}$ can wait after x, if the type 5 timing failures do not exist (i.e., each input can wait longer). Furthermore, if such limited outputs exist only in one module M_i, then we call them *real* limited outputs of M_i. More than one limited output can be real limited as long as they exist in the same module. For example, in Fig. 7 w is a real limited output of M_S^m for $\{M_S^m, M_I\}$ and trace ε. In

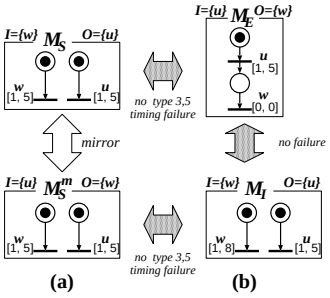

Fig. 8. A correct implementation.

Fig. 8, w is a limited output of M_S^m for $\{M_S^m, M_I\}$ and trace ε, but not a real limited output because u of M_I is also a limited output and it is not in M_S^m.

Definition 4 Given a set $\mathcal{M} = \{M_1, \cdots, M_n\}$ of modules, where $M_k = (\mathcal{I}_k, \mathcal{O}_k, \mathcal{T}_k)$, and another module $M_0 = (\mathcal{I}_0, \mathcal{O}_0, \mathcal{T}_0)$, a *pseudo timing failure* of $\{M_0\} \cup \mathcal{M}$ with respect to M_0 is a finite trace $x = y(w, \tau)$, where $w \in \mathcal{I}_0$, such that $\mathcal{M} \models y$, $M_0 \models x$, and $latest(y, w, M_0) = latest(y, u, M_0)$ holds for some real limited output u of M_0.

In the example of Fig. 7, $(u, 5)$ is a pseudo timing failure of $\{M_S^m, M_I\}$ with respect to M_S^m, because w is a real limited output of M_S^m and has the same latest value as the input u (i.e., $latest(\varepsilon, u, M_S^m) = latest(\varepsilon, w, M_S^m)$ holds). On the other hand, $\{M_S^m, M_I\}$ in Fig. 8 has no pseudo timing failure with respect to M_S^m, because w is not a real limited output. Note that a pseudo timing failure is not included in the type 5 timing failures. However, the existence of pseudo timing failures disallows the correct decision of the conformance by checking failures between system modules and mirror of a specification. This problem can be solved by checking also pseudo timing failures as follows.

Let $\mathsf{failure}P(M_0; \mathcal{M}_C)$ be the union of $\mathsf{failure5}(M_0, \mathcal{M}_C)$ and the set of pseudo timing failures of $\{M_0\} \cup \mathcal{M}_C$ with respect to M_0. Then, we have the following theorem.

Theorem 6 Suppose that the conformance is defined by failure5. $\mathcal{M}_C$ conforms to M_S, iff $\mathsf{failure}P(M_S^m; \mathcal{M}_C) = \emptyset$.

Note that this theorem holds in the examples of Fig. 7 and 8 because $\mathsf{failure}P(M_S^m; M_I) \neq \emptyset$ in the former due to a pseudo timing failure $(u, 5)$ and $\mathsf{failure}P(M_S^m; M_I) = \emptyset$ in the latter. According to this theorem, conformance checking for failure5 can be done by constructing $\mathsf{failure}P(M_S^m; \mathcal{M}_C)$.

V. Conclusion

In this paper, we defined different notions of safety and timing failures for conformance. We compared these definitions and established whether the mirror property holds or not for each of them. Furthermore, we gave additional conditions under which the mirror property holds. Finally, we indicated how to implement a conformance checking procedure for bounded-delay asynchronous circuits even if the mirror property fails.

There are several directions of further work. Firstly, it would be nice to have *logical characterizations* for the various correctness notions and to compare the expressive power of the real-time conformance approach with other methods.

Secondly, the application of our approach to the above mentioned domains needs to be elaborated further. In [12] we applied (a previous version of) a conformance checking algorithm to the verification of bounded delay asynchronous circuits with time Petri nets. Currently we are investigating the conformance checking approach in the context of formal testing based on timed CSP specifications [10].

Thirdly, with real-time constraints the state spaces usually are much bigger than in untimed verification, thus a state explosion problem can arise. The algorithms given in [4, 11] can reduce the state spaces of timed systems, but they use only restricted information about the concurrency between symbols. In [9], we proposed a timed version of the stubborn set method from [3] for model checking of a timed temporal logic and time Petri nets. For conformance checking, in [8] we gave a partial order reduction method for untimed systems similar to the stubborn set method. At the moment we are investigating whether a similar state reduction method, based on the ideas of this paper, can be developed for timed conformance checking.

References

[1] David L. Dill, *Trace theory for automatic hierarchical verification of speed-independent circuits*, MIT press, 1988.

[2] E.M. Clarke, H. Schlingloff, *Model checking*. In: *Handbook of Automated Deduction*, A. Voronkov (ed.), Elsevier, 2000

[3] A. Valmari, *A stubborn attack on state explosion*, Proc. of Workshop on Comupter-Aided Verification, 1990.

[4] T. Rokicki and C. Myers, *Automatic verification of timed circuits*, Computer Aided Verification, LNCS 818 pp.468–480, 1994.

[5] J. Ebergen and R. Berks, *VERDECT: A verifier for Asynchronous Circuits*, IEEE TCCA Newsletter, 1995.

[6] K. L. McMillan, *Trace theoretic verification of asynchronous circuits using unfoldings*, Computer aided verification, LNCS 939, pp. 180–195, 1995.

[7] O. Roig, J. Cortadella and E. Pastor, *Verification of asynchronous circuits by BDD-based model checking of Petri nets*, Application and Theory of Petri Nets 1995, LNCS 935, pp. 374–391, 1995.

[8] T. Yoneda and T. Yoshikawa, *Using partial orders for trace theoretic verification of asynchronous circuits*, Proc. of Second International Symposium on Advanced Research in Asynchronous Circuits and Systems, pp.152-163, 1996.

[9] T. Yoneda and H. Schlingloff, *Efficient verification of parallel real–time systems*, Formal Method in System Design, pp.187–215, 1997.

[10] H. Schlingloff, O. Meyer and T. Hülsing, *Correctness Analysis of an Embedded Controller*, Proc. Int. Conf. on Data Systems in Aerospace (Dasia '99), Lissabon, 1999.

[11] W. Belluomini and C. Myers, *Verification of Timed Systems Using POSETs*, Computer Aided Verification, LNCS 1427, pp.403–415, 1998.

[12] T. Yoneda, B. Zhou and H. Schlingloff, *Verification of bounded delay asynchronous circuits with timed traces*, AMAST'98, LNCS 1548 pp.59–73, 1999.

[13] B. Zhou and T. Yoneda, *Verification of asynchronous circuits with bounded delay model*(in Japanese), IEICE journal, Vol. J82-D-I, No. 7, pp. 819-833, 1999.

[14] T. Yoneda and H. Ryu, *Timed Trace Theoretic Verification Using Partial Order Reduction*, Proc. of Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems, pp. 108–121, 1999.

[15] B. Zhou, T. Yoneda and B. Schlingloff, *Conformance and Mirroring for Timed Asynchronous Circuits*, TIT technical report, 2000.