Formal verification of safety properties in timed circuits [*]

Marco A. Peña[†], Jordi Cortadella[‡], Alex Kondratyev[*] and Enric Pastor[†]

† Department of Computer Architecture
Technical University of Catalonia
08034 Barcelona, Spain
{marcoa, enric}@ac.upc.es

‡ Department of Software
Technical University of Catalonia
08034 Barcelona, Spain
jordic@lsi.upc.es

∗ Theseus Logic Inc.
710 Lakeway Drive, Suite 230
Sunnyvale, CA 94086
alex.kondratyev@theseus.com

Abstract

The incorporation of timing makes circuit verification computationally expensive. This paper proposes a new approach for the verification of timed circuits. Rather than calculating the exact timed state space, a conservative overestimation that fulfills the property under verification is derived. Timing analysis with absolute delays is efficiently performed at the level of event structures and transformed into a set of relative timing constraints.

With this approach, conventional symbolic techniques for reachability analysis can be efficiently combined with timing analysis. Moreover, the set of timing constraints used to prove the correctness of the circuit can also be reported for backannotation purposes. Some preliminary results obtained by a naive implementation of the approach show that systems with more than 10^6 *untimed states can be verified.*

1. Introduction

The correctness of *speed-independent* and *delay-insensitive* circuits can be proved by only considering the sequencing of events and abstracting time as nondeterministic delays. However, the correctness of timed circuits depends on the actual values of event delays. Typically, timing behavior is specified by a set of delays that determine the time duration between the initiation and the completion of an event. This is the valid model for the gates in a circuit, in which gate delays denote the time between the enabledness of the gate and the actual change at the output.

The calculation of the language generated by a timed system is proven to be PSPACE-complete [1], and demonstrated to be highly complex in several contexts such as real-time systems [1, 15] and asynchronous circuits [13, 9, 16, 18, 24, 27]. Difference bounds matrices [5] and decision diagrams [8] have been used to efficiently represent timed polyhedra. Even though these techniques have been combined with partially ordered sets [7], the size of the untimed state space is still the major bottleneck for the analysis of highly concurrent systems.

This paper proposes a novel approach that extends the applicability of the conventional methods based on symbolic reachability analysis to timed circuits. The approach is based on the following observation: the set of traces of a transition system can be covered by a set of marked graphs. Rather than calculating the exact timed state space, our approach performs an *off-line* timing analysis on a set of event structures that covers the traces leading to circuit failures. This timing analysis can be efficiently performed by using McMillan and Dill's algorithm [20]. If some of the failure traces cannot be proven to be timing inconsistent, then the system is incorrect.

The idea of using event structures for timing analysis was already proposed in [17]. However, no algorithm was presented that can handle a general class of transition systems for verification.

The approach presented in this paper not only verifies the circuit, but also provides a set of timing constraints required for its correctness. For speed-independent circuits, the method does not involve any additional overhead with regard to the conventional symbolic methods (e.g. [10]). In [21], an approach with similar goals, but limited to the comparison of circuit paths that start at the same point, was proposed.

With a very naive implementation of a preliminary prototype, circuits with more than 10^6 untimed states have been verified in few minutes of CPU time.

2. Overview

This work presents a formal approach to verify that a circuit with certain timing constraints satisfies a given safety property P. The circuit is modeled by means of a *timed transition system* (TTS), A, composed by an underlying *transition system* (TS), A^-, and two functions, δ^l and δ^u, which associate minimal and maximal delays, respectively, to each event of the system. A given sequence of events of a TTS (a *trace*) is said to be *timing consistent* if it is possible to assign increasing time values to all the events such that their firing times are within the allowed bounds.

The verification problem is posed in terms of the following language inclusion question: $\mathcal{L}(A) \subseteq \mathcal{L}(P)$ [14]. The approach consists in building successive approximations of $\mathcal{L}(A)$ starting from $\mathcal{L}(A^-)$, by adding relative timing constraints [26] in an iterative manner. We start from the TS $A_0 = A^-$ and try to prove the inclusion $\mathcal{L}(A_0) \subseteq \mathcal{L}(P)$ by applying well-known *symbolic model checking* techniques [10, 12]. If this is true, then $\mathcal{L}(A) \subseteq \mathcal{L}(A_0) \subseteq \mathcal{L}(P)$ and A satisfies P without any timing assumption. The verification succeeds.

If P is not satisfied in some state, a trace θ that leads to a failure is generated. If the trace is timing consistent, then the system is incorrect, i.e. violates the required property. However, if the trace is not timing consistent, it can be used to refine the untimed state space and remove other inconsistent traces leading to failure states. To do this, a suffix θ' of the trace θ is taken and an event structure (acyclic marked graph) that covers θ' is built. Timing analysis of the event structure is performed by using the algorithm in [20].

The state space of the event structure is composed with the untimed abstraction of the system A_0, in such a way that at least the wrong trace is removed and no timing consistent trace is removed.

[*]This work has been partially funded by a grant from Intel Corporation, ACiD-WG (ESPRIT 21949), and the Ministry of Education of Spain under contracts CICYTs TIC 98-0410 and TIC 98-0949.

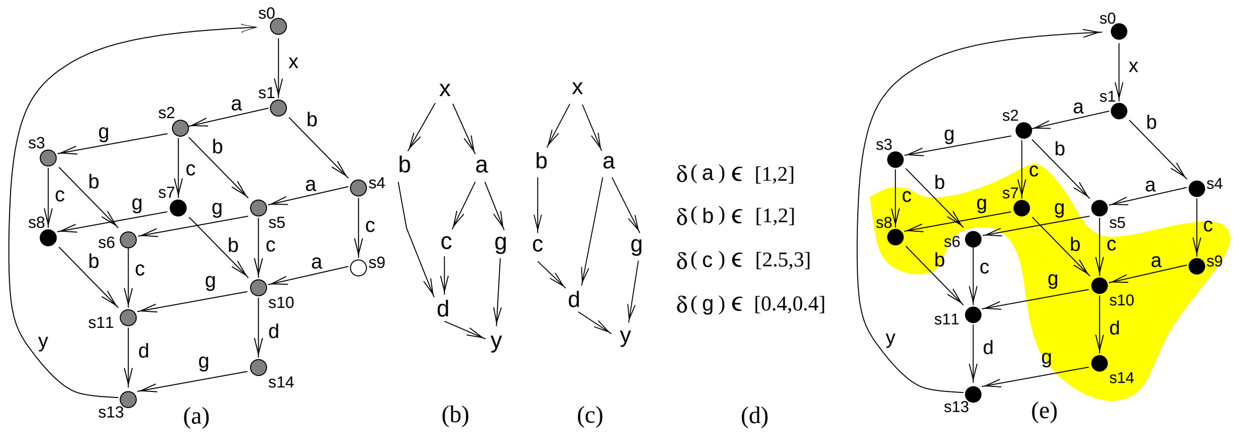

Figure 1. Example 1. (a) Timed transition system with delay intervals specified in (d). (b,c) Event structures covering the traces starting from s_0. (e) Timed state space (shaded states are unreachable).

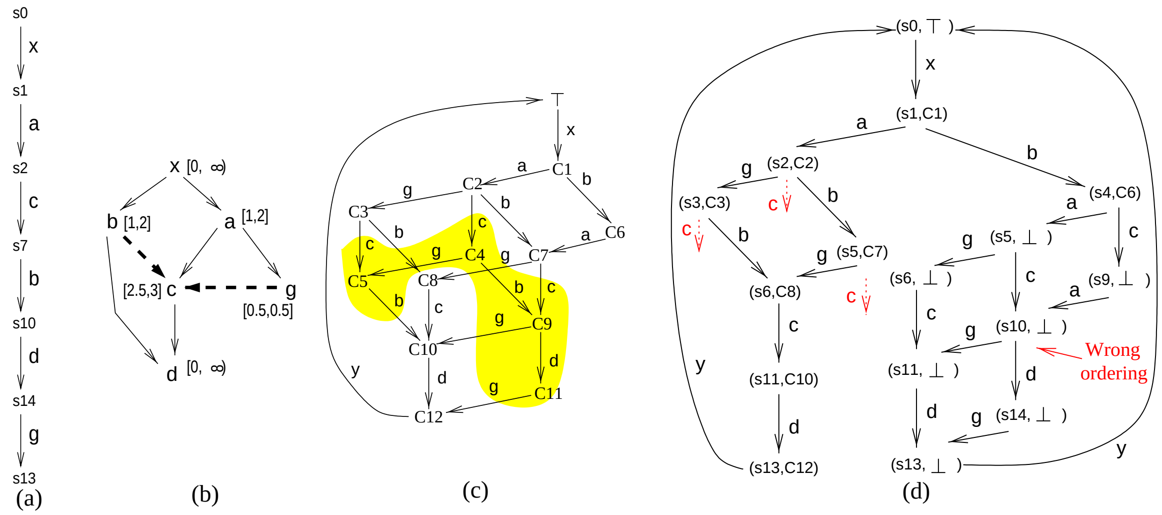

Figure 2. Example 1: first iteration. (a) A wrong trace and its corresponding event structure (b) annotated with timing arcs. (c) State space of the event structure (shaded states are unreachable). (d) TS obtained after composition.

A series of successive approximations A_i of A are constructed iteratively, with containment $\mathcal{L}(A) \subseteq \mathcal{L}(A_i)$ and monotonic convergence, $\mathcal{L}(A_{i+1}) \subseteq \mathcal{L}(A_i)$. At every step $\mathcal{L}(A_i) \subseteq \mathcal{L}(P)$ is checked. Verification stops successfully if the inclusion holds, or fails if a counterexample trace is found.

Iterative approaches for the verification of real-time systems have also been presented in [2, 6]. The major novelty of the approach in this paper is the use of event structures to perform efficient timing analysis, and to incorporate the resulting timing information in the form of relative timing constraints.

2.1. An example

This section illustrates the verification approach by means of a simple example. Figure 1 depicts the TTS modeling a system. Figure 1(a) shows its underlying TS, while Figure 1(d) shows the delay intervals of events a, b, c and g. The delay interval for the rest of events is $[0, \infty)$. Figure 1(e) depicts the state space of the system when the delays are taken into account. A crucial observation is that all traces that start and end at s_0 can be covered by the two event structures depicted in Figures 1(b) and (c). Black states are covered by the event structure (b). White states are covered by the event structure (c). Grey states are covered by both.

Assume that the property to be verified indicates that event g must always precede event d in any possible trace after having visited state s_0. It can be seen that the property holds in the timed state space. However, it does not hold in the untimed state space.

By exploring the state space of Figure 1(a) we see that the property does not hold in state s_{10} if d fires before g. A failure trace from s_0 to s_{10} followed by the firing of d before g can be generated (Figure 2(a)). From this trace, an event structure with the same causality relations can be derived (Figure 2(b)). Note that, in the event structure, c is only triggered by a but not triggered by b. This corresponds to the causal relations derived from the trace, i.e. c is not enabled in s_1 and is enabled after having fired a from s_1.

By timing analysis, we find that b and g always precede c. These timing relations are shown by dotted arcs. Such timing analysis is only valid for the causal relations expressed in the event structure, but it is not valid, for example, in the case when b triggers c. Figure 2(c) depicts the state space of the event structure. Event c is prevented to fire in some states, where its firing would be inconsistent with the timing analysis.

Finally, we incorporate all this information into the system

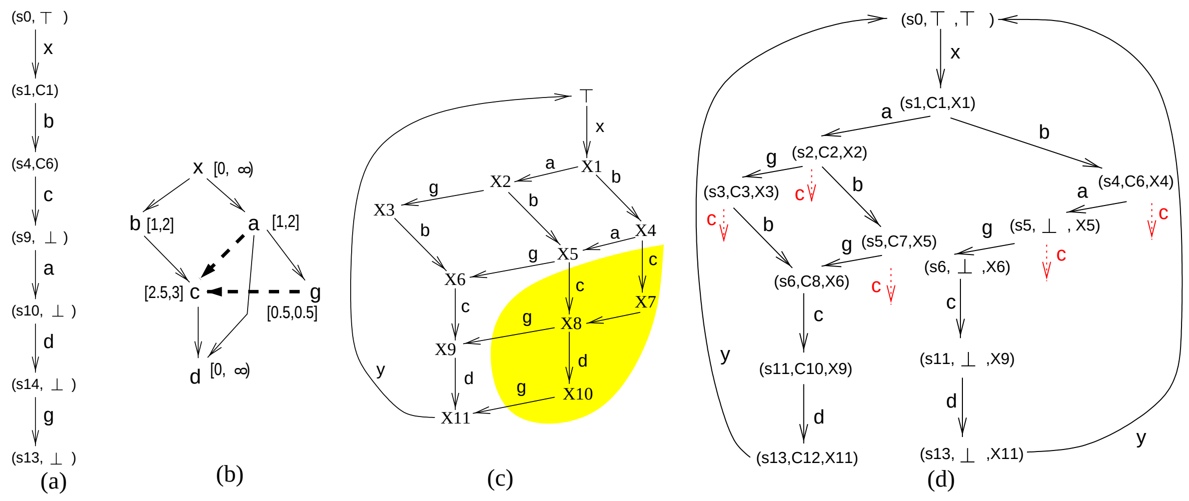

Figure 3. Example 1: second iteration. (a) A wrong trace and its corresponding event structure (b) annotated with timing arcs. (c) State space of the event structure (shaded states are unreachable). (d) TS obtained after composition.

(Figure 2(d)) by composing the original system and the event structure. An event structure being derived from a particular trace gives only partial behaviors of the original system. When the behaviors of the system and the event structure mismatch, the special symbol $\perp$ is used. Some states in the composed system are split into two instances depending on whether they are reached by traces matching (*enabling compatible*) the event structure or not (see states s_5, s_6, s_{11} and s_{13}). Figure 2(d) shows the resulting system. One can easily check that the set of traces is smaller than that of the original system, but larger than that of the actual state space in Figure 1(c), and that only timing inconsistent traces have been removed.

This first step has removed some wrong traces but not all of them. Figure 3 depicts one more refinement. In the resulting system all the wrong traces have been removed, which proves that the system satisfies the property. Although it is not generally true, in this case the final state space contains exactly the same traces than the actual state space shown in Figure 1(c).

The following sections describe the theoretical aspects of the presented approach.

3. Transition systems

This section presents the main models used in the paper. Figure 4 shows the relationship among them.

Definition 3.1 (Transition System) *[4]*
A transition system (TS) *is a quadruple* $A = \langle S, \Sigma, T, s_{in}\rangle$, *where* S *is a non-empty set of* states, Σ *is a non-empty alphabet of* events, $T \subseteq S \times \Sigma \times S$ *is a* transition relation, *and* s_{in} *is the* initial state. *Transitions are denoted by* $s \xrightarrow{e} s'$. *An event* e *is* enabled *at state* s *if* $\exists\, s \xrightarrow{e} s' \in T$. *We will denote by* $\mathcal{E}(s)$ *the set of events enabled at state* s.

Definition 3.2 (Firing region)
Given a TS $A = \langle S, \Sigma, T, s_{in}\rangle$, *the* firing region *of event* e *is defined as* $\mathrm{Fr}(e) = \{s \in S \mid e \in \mathcal{E}(s)\}$.

In the sequel, we will only consider transition systems with the following properties:

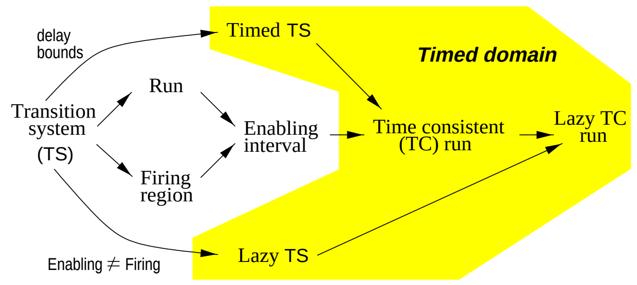

Figure 4. Major notions of Section 3 and their relations.

- S and Σ are finite.
- $s \xrightarrow{e} s' \in T \;\wedge\; s \xrightarrow{e'} s' \in T \;\Rightarrow\; e = e'$ (no multiple arcs between any pair of states).
- $s \xrightarrow{e} s' \in T \;\Rightarrow\; e \notin \mathcal{E}(s')$ (events are *self-disabling*).

Definition 3.3 (Run)
A run *of a* TS $A = \langle S, \Sigma, T, s_{in}\rangle$ *is a sequence* $\sigma = s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} \cdots$, *such that* $s_1 = s_{in}$ *and* $\forall i \geq 1 \;:\; s_i \xrightarrow{e_i} s_{i+1} \in T$. *Event* e_i *is said to* fire *at step* i.

With an abuse of notation, the expressions $s_i \in \sigma$, $s_i \xrightarrow{e_i} s_{i+1} \in \sigma$, $s_i \xrightarrow{e_i} \in \sigma$, $\xrightarrow{e_i} s_{i+1} \in \sigma$, etc, will be often used to denote the fact that different fragments of a sequence belong to a run.

Definition 3.4 (Enabling interval)
Let $A = \langle S, \Sigma, T, s_{in}\rangle$ *be a* TS *and let* $\sigma = s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} \cdots$ *be a run of* A. *Given an event* e *and a state* $s_i \in \sigma$ *such that* $s_i \in \mathrm{Fr}(e)$, $\mathrm{FirstEnabled}(s_i, e)$ *is defined as the state* s_j, $j \leq i$, *such that*

- $j \leq k \leq i \;\Rightarrow\; s_k \in \mathrm{Fr}(e)$ (e *is continuously enabled between* s_j *and* s_i)
- $j > 0 \;\Rightarrow\; s_{j-1} \notin \mathrm{Fr}(e)$ (e *is not enabled before* s_j)

The sequence $s_j \xrightarrow{e_j} \cdots \xrightarrow{e_{i-1}} s_i$ *is called the* enabling interval *of* e *with respect to* s_i.

Time is incorporated to transition systems by assuming that transitions happen instantaneously, while minimal and maximal delay bounds restrict the times at which they may occur.

Definition 3.5 (Timed Transition System) *[15]*
A timed transition system (TTS) *is a triple* $A = \langle A^-, \delta^l, \delta^u \rangle$, *where* $A^- = \langle S, \Sigma, T, \mathsf{s}_{in} \rangle$ *is a* TS *called the* underlying transition system, $\delta^l : \Sigma \rightarrow \Re^+$ *and* $\delta^u : \Sigma \rightarrow \Re^+ \cup \{\infty\}$ *respectively associate a minimal and a maximal* delay bound *to each event, such that* $\forall\, \mathsf{e} \in \Sigma : \delta^l(\mathsf{e}) \leq \delta^u(\mathsf{e})$.

Definition 3.6 (Timing-consistent run)
Let $A = \langle A^-, \delta^l, \delta^u \rangle$ *be a* TTS *and let* $\sigma = \mathsf{s}_1 \xrightarrow{\mathsf{e}_1} \mathsf{s}_2 \xrightarrow{\mathsf{e}_2} \cdots$ *be a run of* A^-. σ *is* timing consistent *with* A *if a sequence* $t_1 t_2 \cdots$ *of real-valued time stamps can be found such that:*

- $t_1 \leq t_2 \leq \cdots$
- $\forall \mathsf{s}_i \xrightarrow{e_i} \mathsf{s}_{i+1} \in \sigma$ *such that* $\mathsf{FirstEnabled}(\mathsf{s}_i, \mathsf{e}_i) = s_j : \delta^l(\mathsf{e}_i) \leq t_{i+1} - t_j \leq \delta^u(\mathsf{e}_i)$
- $\forall \mathsf{s}_i \in \sigma$ *such that* $\mathsf{s}_i \in \mathsf{Fr}(\mathsf{e})$ *and* $\mathsf{FirstEnabled}(\mathsf{s}_i, \mathsf{e}) = s_j : t_i - t_j \leq \delta^u(\mathsf{e})$

The previous definition characterizes those runs which are possible according to the delay bounds of the system. The time stamp t_{i+1} is assigned to state s_{i+1} and corresponds to the *firing time* of event e_i along σ. Similarly, t_j corresponds to the *enabling time*. Thus, the firing time of an event only depends on its enabling time plus certain delay amount within the bounds.

Next, *lazy transition systems* [11] are introduced. The notion of laziness explicitly distinguishes among the enabling and the firing of an event, assuming certain implicit delay between them.

Definition 3.7 (Lazy Transition System)
A lazy transition system (LzTS) *is a five-tuple* $A = \langle S, \Sigma, T, \mathsf{s}_{in}, \mathsf{En} \rangle$, *where* $\langle S, \Sigma, T, \mathsf{s}_{in} \rangle$ *is a* TS, *and the function* $\mathsf{En} : \Sigma \rightarrow 2^S$ *defines the* enabling region *of each event, in such a way that* $\mathsf{Fr}(\mathsf{e}) \subseteq \mathsf{En}(\mathsf{e})$ *for any* $\mathsf{e} \in \Sigma$. *Thus, event* e *is said to be* enabled *at state* $\mathsf{s} \in S$ *if* $\mathsf{s} \in \mathsf{En}(\mathsf{e})$. *An event* e *is said to be* lazy *if* $\mathsf{Fr}(\mathsf{e}) \neq \mathsf{En}(\mathsf{e})$.

Notice that a TS is just a particular case of LzTS in which both enabling and firing regions coincide for all the events. Figure 1(e) shows an example of lazy transition system where event c is lazy since it is enabled in states s_2, s_3, s_4, s_5 and s_6, but is allowed to fire only in s_6.

The notions of $\mathsf{FirstEnabled}(\mathsf{s}_i, \mathsf{e})$ and enabling interval of e with respect to s_i are naturally extended to lazy transition systems from Definition 3.4, by considering $\mathsf{En}(\mathsf{e})$ instead of $\mathsf{Fr}(\mathsf{e})$ for the enabledness of event e.

4. Traces and languages

A common semantics that unifies all the models above can be defined in terms of traces. Based on traces, we will derive several notions that formalize our refinement approach for verification. This flow, depicted in Figure 5, covers the contents of Sections 4 and 5.

We extend the usual notion of trace [19] by associating the set of enabled events to the firing of each event in a run. Thus, each element of the trace keeps track of which events are enabled and which event fires at each step.

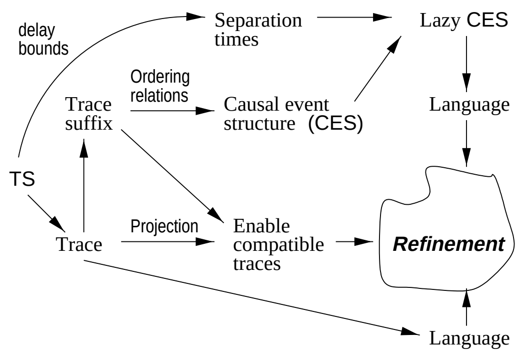

Figure 5. From traces to language refinement.

Definition 4.1 (Trace)
Let Σ *be an alphabet of events. A trace* $\theta = E_1 \xrightarrow{\mathsf{e}_1} E_2 \xrightarrow{\mathsf{e}_2} \cdots$ *is a sequence such that* $\forall i \geq 1 : E_i \subseteq \Sigma$ *and* $\mathsf{e}_i \in E_i$, *where* E_i *denotes the set of events enabled when* e_i *fires.*

Remark: Henceforth, and for the sake of simplicity, all events in a trace will be assumed to be distinct. This assumption can always be enforced by renaming different occurrences of the same event. This renaming does not affect the validity of the theory presented in this paper.

Definition 4.2 (Traces in LzTSs)
Each run $\sigma = \mathsf{s}_1 \xrightarrow{\mathsf{e}_1} \mathsf{s}_2 \xrightarrow{\mathsf{e}_2} \cdots$ *of a* LzTS *defines a trace* $\theta_\sigma = E_1 \xrightarrow{\mathsf{e}_1} E_2 \xrightarrow{\mathsf{e}_2} \cdots$ *where* E_i *is the set of events enabled at* s_i, i.e. $E_i = \mathcal{E}(\mathsf{s}_i)$.

Definition 4.3 (Languages)
The language $\mathcal{L}(A)$ *of a* LzTS A *is the set of traces defined by all runs of* A. *The* language $\mathcal{L}(A)$ *of a* TTS $A = \langle A^-, \delta^l, \delta^u \rangle$ *is the set of traces defined by all timing-consistent runs of* A^-.

Lemma 4.4
Let $A = \langle A^-, \delta^l, \delta^u \rangle$ *be a* TTS. *Then,* $\mathcal{L}(A) \subseteq \mathcal{L}(A^-)$.

The proof directly follows from Definition 4.3.

The following definition is the cornerstone of the verification strategy presented in this paper.

Definition 4.5 (Enabling-compatible trace mapping)
Let $\theta = \cdots \longrightarrow E_0 \xrightarrow{\mathsf{e}_0} E_1 \xrightarrow{\mathsf{e}_1} E_2 \xrightarrow{\mathsf{e}_2} \cdots \xrightarrow{\mathsf{e}_n} E_{n+1} \longrightarrow \cdots$ *be a trace over the alphabet of events* Σ *and let* $\theta' = E'_1 \xrightarrow{\mathsf{e}'_1} E'_2 \xrightarrow{\mathsf{e}'_2} \cdots \xrightarrow{\mathsf{e}'_m} E'_{m+1}$ *be a trace over the alphabet* $\Sigma' \subseteq \Sigma$. *Let* $\theta_t = E_1 \xrightarrow{\mathsf{e}_1} E_2 \xrightarrow{\mathsf{e}_2} \cdots \xrightarrow{\mathsf{e}_n} E_{n+1}$ *be a fragment of* θ. *An* enabling-compatible *mapping of* θ_t *onto* θ' *is a function* $\mathsf{map} : \{E_1, \ldots, E_{n+1}\} \mapsto \{E'_1, \ldots, E'_{m+1}\}$ *such that:*

a) $\mathsf{map}(E_1) = E'_1$ (initialization)

b) $\forall\, 1 \leq i \leq n,\ \mathsf{map}(E_i) = E_i \cap \Sigma'$ (projection)

c) $\forall\, 1 \leq i \leq n,\ (\mathsf{map}(E_i) = \mathsf{map}(E_{i+1})\ \wedge\ \mathsf{e}_i \notin \Sigma')\ \vee$ $(\mathsf{map}(E_i) = E'_j\ \wedge\ \mathsf{map}(E_{i+1}) = E'_{j+1}\ \wedge\ \mathsf{e}_i = \mathsf{e}'_j)$ (firing)

The mapping of θ onto θ' is a function that preserves the enabledness of the events in Σ'. Initially, the events enabled in E'_1 must also be enabled in E_1 (initialization condition). Next, the events of Σ' enabled along θ and θ' must be the same (projection condition). Moreover, θ may fire events that are not relevant to θ'

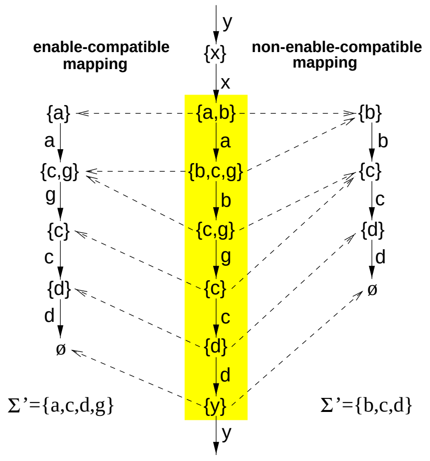

Figure 6. Example of enabling-compatible and non-enabling-compatible mapping.

(when $\mathsf{map}(E_i) = \mathsf{map}(E_{i+1})$ in the firing condition). Since the firing time of an event only depends on its enabling time and its delay (see Section 3), this notion will allow us to apply the timing analysis of θ' to θ in the fragment θ_t.

Figure 6 shows an example of trace mapping of the shadowed fragment. The mapping at the right, with $\Sigma' = \{\mathsf{b},\mathsf{c},\mathsf{d}\}$, is not enabling-compatible since it violates the projection condition when taking $E_i = \{\mathsf{b},\mathsf{c},\mathsf{g}\}$ and $\mathsf{map}(E_i) = \{\mathsf{b}\}$. Clearly, a enables c in θ, whereas c is enabled by b in θ'.

The notions of *enabling interval* and *timing-consistent run* with respect to a pair of functions δ^l and δ^u that assign min/max delays to the events (see Definitions 3.4 and 3.6 respectively), can be naturally extended to traces. From this extension it can be easily proved that a trace defined by a timing-consistent run is also a timing-consistent trace with δ^l and δ^u.

The following is the main theoretical result of this work.

Theorem 4.6 *(see proof in [23])*
Let θ, θ' and θ_t be traces with the same conditions as in Definition 4.5. Let map *be an enabling-compatible mapping from θ_t onto θ'. Let δ^l and δ^u be two functions that assign arbitrary min/max delays to the events in Σ' and 0 and ∞ delays to the events in $\Sigma \setminus \Sigma'$, respectively.*
Then, θ is timing consistent $\iff$ θ' is timing consistent.

The previous theorem states that the timing analysis of a trace can be reduced to the timing analysis of those events that are causally related (events in Σ'). Therefore, the events that are concurrent with all the events of Σ' can be abstracted out. Hence, the timing analysis for one trace can be applied to all those traces that have the same causality relations among the events in Σ'.

5. Event structures

This section presents the basic theory on causal event structures and their traces. Event structures are the only object for which we perform timing analysis, which is rather simple because event structures are acyclic. Taking an event structure which partially specifies the behavior of the original system, we map the timing constraints back to the system behavior by means of composing the system and the event structure.

Definition 5.1 (Causal event structure) *[22]*
A causal event structure (CES), *$CS = \langle \Sigma, \prec \rangle$, is a finite set Σ of* events *and a* precedence relation $\prec \subseteq \Sigma \times \Sigma$ (*irreflexive, antisymmetric and transitive*) *over Σ called the* causality relation.

A causal event structure is usually depicted as a Hasse diagram (Figure 8(a)).

Definition 5.2 (Words and prefixes)
A topological order (*or simply a* word) *of the events of $CS = \langle \Sigma, \prec \rangle$ is a sequence $\mathsf{e}_1 \cdots \mathsf{e}_n \in \Sigma^n$ $(n = \mid \Sigma \mid)$, such that $\forall\ 1 \leq i,j \leq n\ :\ \mathsf{e}_i \prec \mathsf{e}_j \Rightarrow i < j$. Given a word $\sigma = \mathsf{e}_1 \cdots \mathsf{e}_i \mathsf{e}_{i+1} \cdots \mathsf{e}_n$, the i-th prefix of σ is denoted by $\sigma_i = \mathsf{e}_1 \cdots \mathsf{e}_i$. The empty prefix is denoted by σ_0.*

Definition 5.3 (Events enabled by a prefix)
Let $CS = \langle \Sigma, \prec \rangle$ be a CES *and let σ be a word of CS. The set of events enabled by σ_i is defined as $\mathcal{E}(\sigma_i) = \{\mathsf{e}_k \notin \sigma_i \mid \forall \mathsf{e}_j \in \Sigma\ :\ \mathsf{e}_j \prec \mathsf{e}_k \Rightarrow \mathsf{e}_j \in \sigma_i\}$.*

That is, an event e_k is enabled by a prefix σ_i if all the predecessor events (according to $\prec$) are in σ_i but e_k is not.

Definition 5.4 (Traces generated by words)
Let $CS = \langle \Sigma, \prec \rangle$ be a CES *and let $\sigma = \mathsf{e}_1\mathsf{e}_2 \cdots \mathsf{e}_n$ be a word of CS. The trace generated by σ is defined as: $\theta_\sigma = \mathcal{E}(\sigma_0) \xrightarrow{\mathsf{e}_1} \mathcal{E}(\sigma_1) \xrightarrow{\mathsf{e}_2} \cdots \xrightarrow{\mathsf{e}_{n-1}} \mathcal{E}(\sigma_{n-1}) \xrightarrow{\mathsf{e}_n} \emptyset$.*

Definition 5.5 (CES generated by a trace)
Let $\theta = E_1 \xrightarrow{\mathsf{e}_1} E_2 \xrightarrow{\mathsf{e}_2} \cdots \xrightarrow{\mathsf{e}_{n-1}} E_n \xrightarrow{\mathsf{e}_n} E_{n+1}$ be a finite trace. The causal event structure $CS_\theta = \langle \Sigma, \prec \rangle$ generated from θ is defined as follows: $\Sigma = \{\mathsf{e}_1, \ldots, \mathsf{e}_n\}$, $\mathsf{e}_i \prec \mathsf{e}_j \Leftrightarrow i < j \wedge \not\exists E_k \in \theta\ :\ \{\mathsf{e}_i, \mathsf{e}_j\} \subseteq E_k$.

Definition 5.5 is illustrated by Figure 7. Trace (b) is taken from TS (a). The resulting CES, (c), captures the causality relations of the events in the trace. Notice for example, how event c only depends on a according to the trace, although b enables c along other traces.

5.1. Timing analysis on event structures

CESs with timing assumptions can be derived from traces with events annotated with minimum and maximum delay bounds (see Definition 5.5). These assumptions are captured by the notion of *maximal separation time* between the events of a CES. The *maximum separation time* of two events e_1 and e_2 is computed as the maximum difference between their firing times, provided any possible assignment of delays to the events in the graph: $Sep_{max}(\mathsf{e}_1, \mathsf{e}_2) = max\{ft(\mathsf{e}_1) - ft(\mathsf{e}_2) \mid \text{for any delay assignment}\}$, where ft denotes the firing time of the event.

A simple and efficient algorithm for the calculation of the maximal separation between events of a CES can be found in [20]. We can use this information to analyze whether two events are ordered in the time domain, *i.e.* e_1 precedes e_2 if $Sep_{max}(\mathsf{e}_1, \mathsf{e}_2) < 0$.

It is important to point out that the minimum delay bound for all the source events of a CES is conservatively set to 0, given that the prehistory on the enabledness of the events is unknown. With this strategy, timing analysis is still exact in case the CES has only one source event, since the relative firing order of all other events does not depend on the enabling time of their common predecessor.

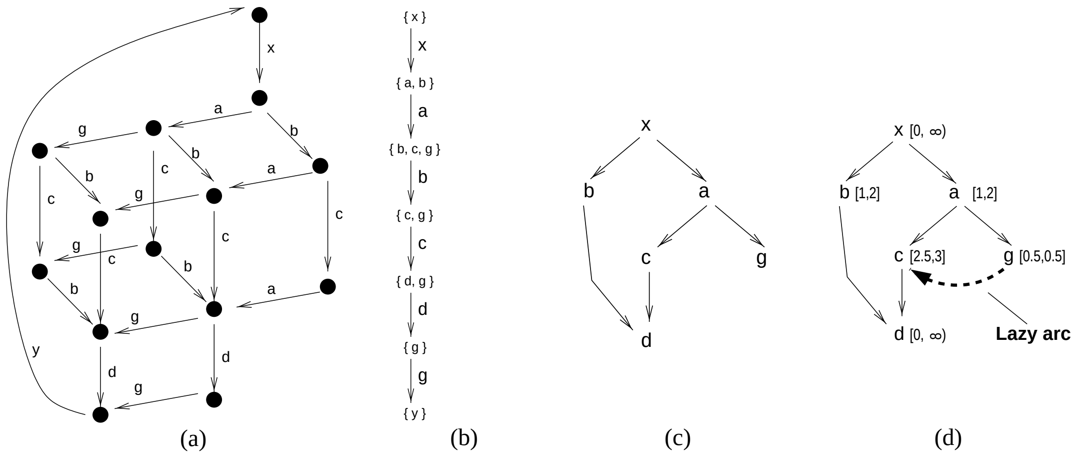

Figure 7. (a) TS, **(b) trace and (c)** CES **obtained from it. (d) Lazy** CES **induced by delay bounds.**

Definition 5.6 (Lazy CES **genenerated by a trace)**
Let $\theta = E_1 \xrightarrow{e_1} \cdots E_n \xrightarrow{e_n} E_{n+1}$ *be a timing-consistent trace of a* TTS $A = \langle A^-, \delta^l, \delta^u \rangle$ *, and let* $CS_\theta = \langle \Sigma, \prec \rangle$*. The pair* $LCS = \langle \Sigma, \prec' \rangle$ *is called a* lazy causal event structure (LzCES)*, where* $\prec' = \prec \cup T$ *, and* $T \subseteq \Sigma \times \Sigma$ *is a set of* lazy causal relations *such that* $T = \{(\mathsf{e}_i, \mathsf{e}_j) \in \Sigma \times \Sigma \mid \mathsf{e}_i \not\prec \mathsf{e}_j \wedge \mathsf{e}_j \not\prec \mathsf{e}_i \wedge Sep_{max}(\mathsf{e}_i, \mathsf{e}_j) < 0\}$

A LzCES coming from certain delay bounds is shown in Figure 7(d) (the lazy arc is depicted by a dashed line).

6. Incorporation of relative timing constraints

This section describes how to refine the set of traces produced by a lazy TS by considering the timing constraints coming from event delay bounds. The timing constraints are derived by the analysis of a causal event structure corresponding to an eligible trace of a lazy TS in the untimed domain. The refinement is performed through the parallel composition of a LzTS and a LzCES. Defining the parallel composition requires both descriptions to be represented in a uniform way. To satisfy this requirement we first introduce a state-based representation for CESs.

6.1. State-based representation of a CES

Form a causal event structure one can obtain an underlying transition system. This process relies on the notion of *configuration*, which plays the role of global state.

Definition 6.1 (Configuration)
Let $CS = \langle \Sigma, \prec \rangle$ *be a* CES. $\mathcal{C} \subseteq \Sigma$ *is a* configuration *iff* $\mathcal{C}$ *is left-closed,* i.e. $\forall \mathsf{a} \in \mathcal{C}$ *all predecessors of* a *by* $\prec$ *are in* $\mathcal{C}$*.*

Clearly, every prefix σ_i of word a σ in CES is left-closed and hence defines a configuration which is reached by firing the events from σ_i. Consideration of all possible words and their prefixes gives the *set of reachable configurations*, C, where the initial configuration due to the empty prefix σ_0 is denoted by $\top$. The set of reachable configurations together with a partial order $\subset$ defines a *graph of reachable configurations*.

Definition 6.2 (Graph of reachable configurations)
The graph of reachable configurations for CES $CS = \langle \Sigma, \prec \rangle$ *is a Hasse diagram over the set of reachable configurations of* CS *and the partial order* $\subset$ *interpreted in set-theoretical sense.*

For the general case of a LzCES, $LCS = \langle \Sigma, \prec \rangle$, the graph of reachable configurations can be modeled by a LzTS $G = \langle C, \Sigma, T, \top, \mathsf{En} \rangle$ where $\mathcal{C}_1 \xrightarrow{\mathsf{e}} \mathcal{C}_2 \in T$ iff $\mathcal{C}_2$ is reached by firing e from $\mathcal{C}_1$, and $\mathsf{En}(\mathsf{e}) = \{\mathcal{C} \in C \mid \mathsf{e} \in \mathcal{E}(\mathcal{C})\}$. An example of a graph of reachable configurations is shown in Figure 8(c). In this graph every arc $(\mathcal{C}_1, \mathcal{C}_2)$ is attributed by an event which expands configuration $\mathcal{C}_1$ into $\mathcal{C}_2$ (the firing event).

The following statement shows that in CES a configuration is uniquely defined by the set of enabled events.

Theorem 6.3 (Configurations and enablings) *[23]*
Any pair of configurations $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *(*$\mathcal{C}_1 \neq \mathcal{C}_2$*) of a* CES $CS = \langle \Sigma, \prec \rangle$ *has different sets* $\mathcal{E}(\mathcal{C}_1)$ *and* $\mathcal{E}(\mathcal{C}_2)$ *of enabled events, i.e.* $\mathcal{C}_1 \neq \mathcal{C}_2 \Rightarrow \mathcal{E}(\mathcal{C}_1) \neq \mathcal{E}(\mathcal{C}_2)$.

In the sequel we will indistinctly use configurations or their enablings to characterize the states of a CES. Based on this one-to-one correspondence instead of a graph of reachable configurations one could consider an isomorphic *graph of reachable enablings* (Figure 8(c)).

6.2. Refining the reachability space by timing constraints

At this moment we have two objects at hands: a lazy TS A, and another lazy TS G obtained from an event structure CS_θ . CS_θ is derived by a particular trace θ of A (actually by an appropriate suffix, see Figure 5), thus giving only a partial specification of the behavior of A. CS_θ is refined through the exact timing analysis yielding the lazy TS G.

Refining the behavior of A by the timing constraints incorporated in G can be done by calculating the *enabling-compatible product* of G and A, which is a particular case of transition system product under the restrictions of making synchronization by the *same transitions* and the *same enabling conditions*.

For sake of simplicity, before introducing the product rules we will add the special configuration $\perp$ to G. $\perp$ denotes the fact that the product is not synchronizing with the state space of the CES and, therefore, no timing analysis is applied for the involved traces.

The enabling-compatible product of $A = \langle S, \Sigma_A, T_A, \mathsf{s}_{in}, \mathsf{En}_A \rangle$, and $G = \langle C \cup \perp, \Sigma_G, T_G, \top, \mathsf{En}_G \rangle$ with $\Sigma_G \subseteq \Sigma_A$ is a new LzTS $\langle S', \Sigma_A, T', \mathsf{s}'_{in}, \mathsf{En}' \rangle$ where:

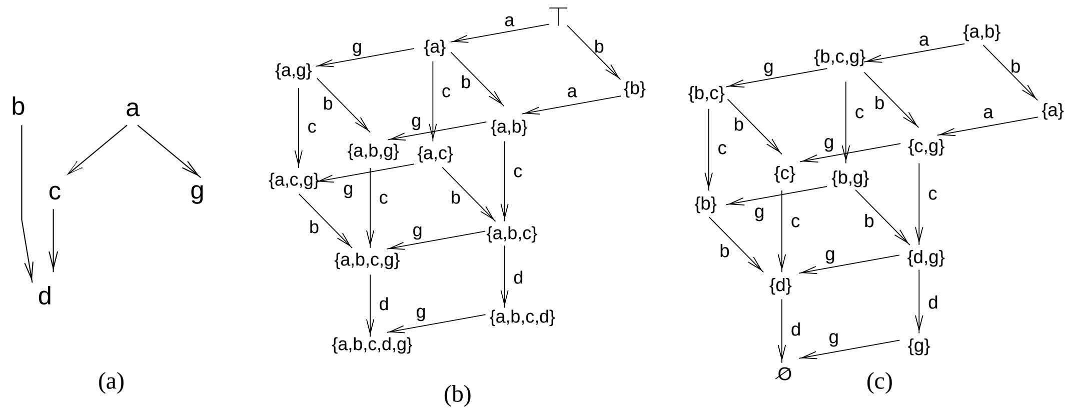

Figure 8. (a) Causal event structure. Graph of reachable configurations (b) and enablings (c).

- $S' \subseteq S \times (C \cup \bot)$,
- $\mathsf{s}'_{in} = (\mathsf{s}_{in}, \top)$ if $\mathcal{E}(\top) \subseteq \mathcal{E}(\mathsf{s}_{in})$, and $\mathsf{s}'_{in} = (\mathsf{s}_{in}, \bot)$ otherwise, and
- $\forall \mathsf{e} \in \Sigma_A,\ \mathsf{En}'(\mathsf{e}) = \{(\mathsf{s}, \mathcal{C}) \in S' \mid \mathsf{s} \in \mathsf{En}_A(\mathsf{e})\}$.

The transition relation T' is defined by the rules below. These are implied by the conditions of Definition 4.5 on enabling compatibility of traces. The fact that $(\mathsf{s}, \mathcal{C}) \in S'$ denotes that s and $\mathcal{C}$ have been reached by prefixes that are enabling compatible, and that $\mathsf{map}(\mathcal{E}(\mathsf{s})) = \mathcal{E}(\mathcal{C})$. Given a state $(\mathsf{s}, \mathcal{C})$ with $\mathcal{C} \neq \bot$, we will say that the state is in the timed domain, indicating that the timing analysis performed on CS_θ can be applied to s.

Transitions entering the timed domain

Transition	Conditions
$(\mathsf{s}, \bot) \xrightarrow{\mathsf{e}} (\mathsf{s}', \top)$	$\mathsf{enter} \equiv \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T_A \wedge$ $\mathcal{E}(\top) \subseteq \mathcal{E}(\mathsf{s}') \cap \Sigma_G$

These transitions are fired when the events enabled in $\top$ are also enabled in s'. Thus, timing analysis can start being applied from $(\mathsf{s}', \top)$.

Staying inside the timed domain

Transition	Conditions
$(\mathsf{s}, \mathcal{C}) \xrightarrow{\mathsf{e}} (\mathsf{s}', \mathcal{C})$	$\mathsf{inside1} \equiv \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T_A \wedge$ $\mathcal{E}(\mathsf{s}) \cap \Sigma_G = \mathcal{E}(\mathsf{s}') \cap \Sigma_G$
$(\mathsf{s}, \mathcal{C}) \xrightarrow{\mathsf{e}} (\mathsf{s}', \mathcal{C}')$	$\mathsf{inside2} \equiv \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T_A \wedge$ $\mathcal{C} \xrightarrow{\mathsf{e}} \mathcal{C}' \in T_G \wedge \mathcal{E}(\mathsf{s}') \cap \Sigma_G = \mathcal{E}(\mathcal{C}')$

Inside1 corresponds to the condition in which e does not synchronize with G. Here the enablings of state $\mathcal{C}$ must be preserved, i.e. the firing of e cannot disable or enable events in Σ_G.

For inside2, both A and G make a synchronized move which might affect the events from Σ_G in exactly the same way: if $\mathsf{a} \in \Sigma_G$ becomes enabled in A due to this move, it should also become enabled in G, and vice versa.

Exiting or staying outside the timed domain

Transition	Conditions
$(\mathsf{s}, \mathcal{C}) \xrightarrow{\mathsf{e}} (\mathsf{s}', \bot)$	$\mathsf{exit} \equiv \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T_A \wedge$ $\neg(\mathsf{enter} \vee \mathsf{inside1} \vee \mathsf{inside2})$

It can be shown that, in the enabling-compatible product, only the traces of the original TS which are enabling compatible with the event structure are refined. This refinement excludes the traces which are timing inconsistent with respect to timing constraints coming from the event structure. All other traces are not changed, thus guaranteeing the conservativeness of the approach.

7. Algorithm for timed verification

7.1. Timing refinement

The verification problem can be solved by checking the language inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(P)$ of a timed system described by a TTS A and a property described by P .

Figure 9 shows the timed verification procedure. The function *untimed_verification* checks whether a trace violating P is present in A' . If such a trace exists, a finite prefix, θ, demonstrating the wrong behavior is returned. This prefix is checked for timing inconsistency by building and analyzing the corresponding event structure (procedure *build_event_structure*. If no event structure can disprove the feasibility of the trace θ the verification returns θ as an example of violation of P. Otherwise the system is refined through the composition with an event structure.

The *timed_verification* procedure does not depend on any particular implementation of the *untimed_verification* function. We have implemented, however, an approach based on efficient symbolic model checking techniques [10]. Basically, we explore A' looking for failure states where P is violated. Then, a backward traversal is performed to generate a trace, leading from the initial state to the failure, reproducing the discrepancy with P.

7.2. Timing analysis of failures

Given the trace θ generated by the function *untimed_verification*, we search for the shortest suffix θ'' ($\theta = \theta' \cdot \theta''$) such that θ'' is timing inconsistent with the delay bounds δ^l and δ^u . Let us illustrate the process by means of an example.

Consider the trace $\{\mathsf{x}\} \xrightarrow{\mathsf{x}} \{\mathsf{a}, \mathsf{b}\} \xrightarrow{\mathsf{a}} \{\mathsf{b}, \mathsf{c}, \mathsf{g}\} \xrightarrow{\mathsf{c}} \{\mathsf{b}, \mathsf{g}\} \xrightarrow{\mathsf{b}} \{\mathsf{d}, \mathsf{g}\} \xrightarrow{\mathsf{d}} \{\mathsf{g}\} \xrightarrow{\mathsf{g}} \{\mathsf{y}\}$ for the TS of Figure 1(a) and assume the delay bounds specified in Figure 1(d). Recall that in this example, the property being verified says that g must always fire before d.

The shortest possible suffix is given by the trace $\{\mathsf{b}, \mathsf{g}\} \xrightarrow{\mathsf{b}} \{\mathsf{d}, \mathsf{g}\} \xrightarrow{\mathsf{d}} \{\mathsf{g}\} \xrightarrow{\mathsf{g}} \{\mathsf{y}\}$, from which the event structure of Figure 10(a) is derived, according to Definition 5.5.

```
function timed_verification ( $A = \langle S_A, \Sigma_A, T_A, \mathsf{s}_{in\,A}, \delta^l, \delta^u \rangle, P$ )
    $A' = \langle S_A, \Sigma_A, T_A, \mathsf{s}_{in\,A} \rangle$ ;
    repeat
        $\theta$ := untimed_verification($A'$, $P$);
        if (empty $\theta$) return(SUCCESS);
        $LCS$ := build_event_structure($A'$, $\theta$, $\delta^l$, $\delta^u$);
        if (empty $LCS$) return(FAIL, $\theta$);
        $A''$ := compose($A'$, $LCS$);
        $A'$ := $A''$;
    end repeat
end function
function build_event_structure ( $A' = \langle S, \Sigma, T, \mathsf{s}_{in} \rangle$, $\theta$, $\delta^l$, $\delta^u$ )
    $\theta''$ := shortest_suffix($\theta$);
    repeat
        $\theta''$ := add_predecessor($\theta''$, $\theta$);
        $CS$ := build_event_structure($A'$, $\theta''$);
        if (timing_consistent($CS$, $\delta^l$, $\delta^u$))
            $L$ := compute_lazy_arcs($CS$, $\delta^l$, $\delta^u$);
            $LCS$ := add_lazy_arcs($CS$, $L$);
            return ($LCS$);
        end if
    while ($\theta'' \neq \theta$);
    return (empty $CS$);
end function
```

Figure 9. Algorithms described in Section 7.

The timing analysis can conclude nothing about the occurrence order of events d and g, since both can fire concurrently. Therefore the algorithm continues by moving one step backwards along the trace and repeats the same process again. Figure 10 depicts the three attempts needed to find the shortest sufficient suffix of the original trace. According to it, timing analysis concludes that b and g occur before c (and consequently before d). This is shown by the dashed arcs in the lazy event structure of Figure 10(c).

The function *build_event_structure* builds the shortest suffix θ'' of the trace returned by the untimed verification procedure such that the timing analysis shows a timing inconsistency with the delays imposed by A. An *event structure* CS is constructed by using the causal relations of the events in θ''.

Function *timing_consistent* performs timing analysis over the CS. It implements the algorithm described in [20] for timing analysis of an acyclic graph of events with min/max delay constraints.

If the trace is not timing consistent, function *compute_lazy_arcs* also extracts a set of *relative timing* constraints from CS, *i.e.* a set of additional orderings between the events of θ'' imposed by the delay bounds. These new constraints are added to the initial CS as *lazy arcs* by function *add_lazy_arcs*. The resulting *lazy event structure* LCS models only those orderings of the events of θ'' which are timing consistent with the delays imposed by A.

7.3. Incorporation of constraints

Finally, we develop the composition algorithm that implements the enabling-compatible product between A' and a lazy CES, LCS, which derives A'' by removing from A' all traces contradicting the timing orderings of events in LCS. Thus $\mathcal{L}(A'') \subseteq \mathcal{L}(A')$. The resulting A'' is a new lazy TS where:

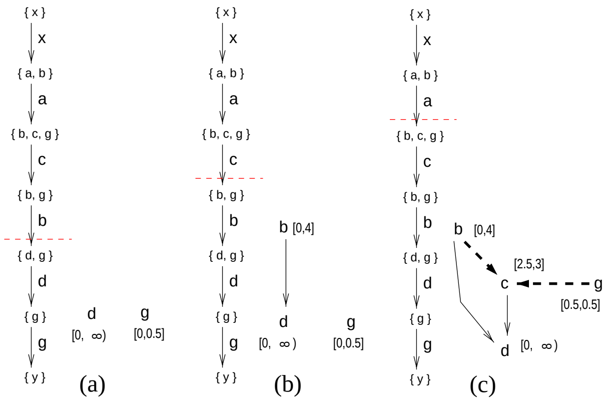

Figure 10. Example 1: generation of the shortest suffix of the trace depicted in Figure 1(a), and corresponding event structures. Three steps are needed.

- The state space may be split in two parts: one following the enabling orders of the events in LCS, and the other one where the enablings are not followed. The former corresponds to the state subspace where the constraints imposed by LCS apply (the timed subspace). In the latter, LCS does not apply (the untimed subspace).
- In the timed subspace, some events are prevented to fire when they are enabled. More precisely, the composition with LCS allows only those firing orderings which are consistent with the timing analysis.

7.4. Correctness

The correctness of the *timed_verification* algorithm is guaranteed by the following facts:

(i) The language of the TTS being verified is a subset of the language of the initial untimed abstraction (its underlying TS). This condition is proved by Lemma 4.4.

(ii) Conservativeness: the *compose* function does not remove any trace which is timing consistent with the delays δ^l and δ^u of the verified TTS. This is guaranteed by the composition rules of the enabling-compatible product (see Section 6.2).

(iii) Convergence: for a particular class of systems the verification requires only few refinements to converge (more details in next section). For the general class of systems a pre-defined upper bound on the number of refinements can be imposed. This could produce false negatives during verification. However it is in full correspondence to the conservative nature of the suggested approach.

7.5. Convergence

Each composition step of the original LzTS A' with the lazy event structure LCS implicitly performs an unfolding of A' separating traces that are enabling compatible with LCS and those which are not.

The convergence of the refinement procedure for the class of Marked Graphs is guaranteed by the known results on termination of separation times analysis in a finite number of unfolding iterations [16]. Nevertheless the upper bound on the number of iterations could be quite high (depends on the ratio of critical and sub-critical cycles). This is an inherent limitation of exact separation analysis and, for practical applications, it is better to work with pre-established separation bounds and do not unfold beyond those bounds. Though it gives only conservative verification, an acceptance of pre-defined upper bounds seems to be a reasonable option because the largest class of systems for which the separation times analysis could be performed exactly are free and unique choice systems [16] (beyond them the calculation of separation times is inherently conservative).

However there is an important practical class of systems for which the refinement procedure is especially simple and is exact for few unfolding iterations. The characterization of this class is done in terms of the so-called *nodal states*.

Definition 7.1 (Nodal state)
A state s *of a* TS *is called nodal if* $\forall \mathsf{s}', \mathsf{s}' \xrightarrow{\mathsf{e}} \mathsf{s}$, $|\mathcal{E}(\mathsf{s}')| = 1$.

Definition 7.1 points that all direct predecessors of a nodal state are synchronized in that state, *i.e.* to the moment when a system arrives to a nodal state all concurrent activities have been finished.

Nodal states are natural points from which the timing analysis is convenient to start. Any event enabled somewhere in a path to a nodal state must fire before reaching this state and, hence, timing analysis from a nodal state does not depend on the prehistory of the process behavior. We will call a TS in which every trace passes through at least one nodal state as *strongly synchronized*. Note that the requirement of breaking traces by a *set* of nodal states is essential here because it is easy to construct an example of TS with choices, in which different branches of a choice would have different nodal states and none of them could serve as a "global synchronizer" for the whole TS. For strongly synchronized TSs, timing analysis can always be performed on event structures with only one source node. Thus, timing analysis can be exact in those cases (see section 5.1).

In the TS in Figure 1, states s_0, s_1 and s_{13} are nodal, this TS has no conflicting events (no choice) and therefore each nodal state is a "global synchronizer" because it breaks all the TS cycles.

In a strongly synchronized TS, given a faulty trace θ with an "improper" ordering of the pair of events a and c, checking the timing consistency by a and c might be reduced to the analysis of the suffix θ_t starting from the nodal state closest to a and c.

By θ_t one can construct the corresponding CES to check whether a and c might occur in the order they have in θ. However in case of cyclic behavior, θ might continue in such a way that the first n occurrences of events a and c satisfy the checked properties while their $n+1$ occurrences have an "improper" ordering. The nice feature of strongly synchronized TS is that timing analysis made for trace θ can be equally applied for "later" occurrences of a and c because the analysis, started at a nodal state, does not depend on prehistory. Therefore timing inconsistency of θ implies also timing inconsistency for any cyclic unfolding of θ, from which it immediately follows the exactness and convergence of the suggested procedure for verification.

The practical significance of the class of strongly synchronized TS could be shown by analyzing the known set of asynchronous benchmarks: more than 80% of the specifications are strongly synchronized.

Beyond the class of strongly synchronized TSs our verification procedure would be conservative in general. Still in many cases it might require just few iterations in unfolding the TS to reach the exact separation analysis. For example, [3] shows the fast convergence of separation times analysis for pipelined specifications, which are inherently not strongly synchronized.

name	Σ	S	G	S_u	S_f	TC	C	cpu
sbf-rd-ctl	8(5)	19	10	74	16	4	Y	2
rcv-setup	5(2)	14	6	78	34	2	N	1
alloc-outbnd	9(5)	21	11	82	20	4	Y	3
ebergen	5(3)	18	9	83	22	1	N	1
mp-fwd-pkt	8(5)	22	8	186	70	8	Y	5
dff	3(1)	14	6	255	164	6	N	3
half	4(2)	14	7	227	133	1	N	1
chu133	7(4)	24	9	288	204	2	N	1
converta	5(3)	18	12	408	244	10	N	12
nowick	6(3)	20	10	510	292	4	Y	3
chu150	6(3)	26	8	520	339	3	N	1
sbuf-snd-ctl	8(5)	27	13	1592	1081	18	N	54
vme	6(3)	24	12	1736	1460	21	Y	30
rpdtf	5(1)	22	8	2612	1841	2	N	2
tsend-bm	9(4)	40	12	3880	2999	3	N	46
sbf-snd-pkt2	9(5)	28	13	4544	4044	19	Y	103
sbf-ram-wrt	12(7)	64	15	14016	12362	34	N	415
ram-rd-sbuf	11(6)	39	16	19328	17488	36	Y	550
mr1	9(5)	190	16	21076	11574	29	Y	317
mr0	11(6)	302	20	727304	642291	2	N	48
trimos-send	9(6)	336	24	2.1 e6	1.8 e6	1	N	127
mmu	8(4)	174	22	5.6 e6	5.2 e6	3	N	480

Table 1. Experimental results for the verification of absence of hazards in asynchronous circuits.

8. Experimental results

A naive prototype of the proposed method has been implemented by using state-of-the-art symbolic BDD-based techniques for reachability analysis.

The following experiments have been performed on a set of specifications given as *Signal Transition Graphs*. A speed-independent complex-gate implementation (i.e. one complex gate per output) of each specification has been obtained by using `petrify`. The complex gates have been decomposed and mapped into a library with only 2-input gates (NAND2, NOR2 and inverter). Conventional decomposition for synchronous circuits has been used for technology mapping (`map` command in SIS). None of the examples were hazard-free under the unbounded delay model after decomposition.

Next, the following interval delays were assigned to the events of the system: $[0.9, 1.1]$ for inverters, $[1.35, 1.65]$ for NAND2/NOR2 gates and $[9, 11]$ for the events produced by the environment. The property verified for each circuit was absence of hazards with the given delays. Formally, the property was modeled as semi-modularity for each event. The composition of the environment with the circuit defines the transition system that must be verified.

Although most specifications were marked graphs (choice-free Petri nets), the transition system obtained after the composition with the circuit manifested a great variety of causality relations among the events (OR, AND and complex combinations of both) produced by the funcionality of the gates.

Table 1 reports the obtained results. Columns Σ and S contain the total number of signals (outputs are shown in parenthesis) and states of the specification. Columns G and S_u indicate the number of gates and the number of untimed states of the circuit.

Column S_f indicates the number of untimed failure states (only failure states reachable from non-failure states are generated). The column TC indicates the number of event structures (timing constraints) generated for timing analysis. This corresponds to the number of iterations of the algorithm. The column C indicates whether the circuit is correct or not. Finally, CPU times are given in seconds.

The results show that, even with a naive implementation, systems with more than 10^6 untimed states could be verified. The computational cost of the algorithm highly depends on the number of timing constraints required to refine the untimed state space. Some heuristics to improve the strategies to select adequate event structures will be explored in the future.

The three largest examples were proved to be hazardous. Only few iterations were required to find an erroneous trace. On the other hand, some circuits required a lot of timing constraints to prove its correctness (e.g. `ram-rd-sbf` and `mr1`). We believe that many of these constraints can be redundant and simplified when considering the complete set of constraints as a whole.

In the future we intend to implement clever strategies for state encoding, variable ordering, state traversal, etc, that should produce tangible improvements on the size of the systems that will be handled by the tool. We also want to explore strategies to simplify the set of constraints required for correctness.

9. Conclusions

We believe that the conventional symbolic methods for reachability analysis can be efficiently extended for timed systems. In this paper we have presented the basic theory and some preliminary experiments for the verification of timed circuits.

Even though timed systems have an inherent delay for each event, it is evident that, in practice, many of the timing constraints imposed by these delays are not required for the correctness of a system. The proposed approach is not only useful to verify correctness, but also to backannotate the set of timing constraints that have been used to prove it. These constraints correspond to the timing arcs derived from the analysis of event structures. This information is crucial in frameworks in which synthesis and verification are iteratively invoked to design systems that must meet functional and non-functional constraints.

The fact that timing analysis is performed by event structures, while verification can be performed in systems with any type of causality relation, opens the door to symbolic analysis. This can be achieved by using techniques similar to those proposed in [3] and based on quantifier-free Presburger arithmetic [25].

References

[1] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] R. Alur, A. Itai, R.P. Kurshan, and M. Yannakakis. Timing Verification by Successive Approximation. *Information and Computation*, 118(1):142–157, 1995.

[3] T. Amon and H. Hulgaard. Symbolic time separation of events. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 83–93, April 1999.

[4] A. Arnold. *Finite Transition Systems*. Prentice-Hall, 1994.

[5] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data structures for the verification of timed automata. In O. Maler, editor, *Proc. HART'97*, volume 1201 of *LNCS*, pages 346–360. Springer-Verlag, 1997.

[6] F. Balarin and A.L. Sangiovanni-Vincentelli. An iterative approach to verification of real-time systems. *Formal Methods in System Design*, 6:67–95, January 1995.

[7] W. Belluomini and C.J. Myers. Verification of timed systems using POSETs. In *Proc. International Conference on Computer Aided Verification*, pages 403–415, 1998.

[8] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In O. Grumberg, editor, *Computer-Aided Verification'97*, volume 1254 of *LNCS*, pages 179–190. Springer-Verlag, 1997.

[9] J. R. Burch. Delay models for verifying speed-dependent asynchronous circuits. In *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, March 1992.

[10] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. of the Fifth Annual Symposium on Logic in Computer Science*, June 1990.

[11] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Lazy transition systems: application to timing optimization of asynchronous circuits. In *Proc. ICCAD*, November 1998.

[12] O. Coudert, J. C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. *Computer-Aided Verification'90*, pages 75–84, 1990.

[13] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Verification of asynchronous interface circuits with bounded wire delays. In *Proc. ICCAD*, pages 188–195, November 1992.

[14] A. Gupta. Formal Hardware Verification Methods: A Survey. *Formal Methods in System Design*, 1:151–238, 1992.

[15] T. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proc. of the 18th ACM Symposium on Principles of Programming Languages*, pages 353–366, 1991.

[16] H. Hulgaard and S. M. Burns. Bounded delay timing analysis of a class of CSP programs with choice. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 2–11, November 1994.

[17] H. Kim and P.A. Beerel. Relative timing based verification of timed circuits and systems. In *International Workshop on Logic Synthesis*, June 1999.

[18] O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In P.E. Camurati and H. Eveking, editors, *Proc. CHARME'95*, volume 987 of *LNCS*, pages 189–205. Springer-Verlag, 1995.

[19] A. Mazurkiewicz. Basic notions of trace theory. In J. W. Baker, W. P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 285–363. Springer-Verlag, 1988.

[20] K. L. McMillan and D. L. Dill. Algorithms for interface verification. In *Proc. of the International Conference on Computer Design*, October 1992.

[21] R. Negulescu and A. Peeters. Verification of speed-dependences in single-rail handshake circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, March 1998.

[22] M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains. *Theoretical Computer Science*, 13:85–108, 1981.

[23] M.A. Peña, J. Cortadella, A. Kondratyev, and E. Pastor. Formal verification of safety properties in timed circuits. Technical Report RR-99/49 http://www.ac.upc.es/homes/marcoa/TechReports.html, UPC/DAC, October 1999.

[24] A. Semenov and A. Yakovlev. Verification of asynchronous circuits using time Petri-net unfolding. In *Proc. Design Automation Conference*, 1996.

[25] R.E. Shostack. A practical decision procedure for arithmetic with function symbols. *Journal of the ACM*, 26(2):351–360, April 1979.

[26] K. Stevens, R. Ginosar, and S. Rotem. Relative timing. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 208–218, April 1999.

[27] E. Verlin, G. de Jong, and B. Lin. Efficient partial enumeration for timing analysis of asynchronous systems. In *Proc. Design Automation Conference*, 1996.